1. **Enumerations:**
   - Enumerators contain a list of constant values that apply to a certain type of data, or object.
   - They can be useful in setting a scope of values for a particular object.
   - An enumeration defines a class type.
   - An enumeration can have constructors, methods, and instance variables.
   - An enum is actually a new type of class.
   - You can declare them as inner classes or outer classes.
   - You can declare variables of an enum type.
   - Each declared value is an instance of the enum class.
   - Enums are implicitly public, static, and final.
   - enums extend java.lang.Enum and implement java.lang.Comparable.
   - Supports **equals, "==", compareTo, ordinal,** etc.
   - Enums override toString() and provide valueOf(é), name().

Points to remember for Java Enum

- enum improves type safety
- enum can be easily used in switch
- enum can be traversed
- enum can have fields, constructors and methods
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

1.1 Enumeration fundamentals :
enumeration is a special kind of class that includes a list of constant values. The values in the enumeration list define the values an object can have

Creating Enumerations

- When creating an enumeration list, we don't use the keyword class and when you create a enumeration object, we don't use the keyword new
- To create an enumeration list we need to import **java.util.Enumeration**
- An enumeration is created using the **enum** keyword followed by the variable Name we want associated with the list

Syntax:

```
public enum variableName{
   ITEM1
```

```
 public static void main( String args[] )
 {
   Gender s=Gender.FEMALE;
  if(s==Gender.MALE)
                         System.out.println("Both are not equal");


           }

           }
```

program 2: To find Smallest of given number.

```
enum Value {
                         a(10), b(20);

                         int a1;
                         int getValue(){ return a1;}
                         Value(int value)
                         {
                                     this.a1=value;
                         }
           }

class Enu
{
 public static void main( String args[] )
 {
   int s=Value.a.getValue();

  if(s<Value.b.getValue())
              System.out.println("a value is small");
           }

           }
```

2)if else: The Java if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block* is executed.

Syntax: if(condition) //code if condition is true

else

```
 //code if condition is false
 }
```

program 1:

```
enum Gender {
                         MALE, FEMALE, UNKNOWN;
```

```
            }

class cont
{
 public static void main( String args[] )
 {
   Gender s=Gender.FEMALE;
  if(s==Gender.MALE)
             System.out.println("both are equal");
  else
             System.out.println("Both are not equal");

  }
 }
```
output: Both are not equal

Program 2:To find Smallest of two numbers

```
 enum Value {
                         a(10), b(20);

                         int a1;
                         int getValue(){ return a1;}
                         Value(int value)
                         {
                                     this.a1=value;
                         }
             }

class Enu
{
 public static void main( String args[] )
 {
   int s=Value.a.getValue();

  if(s<Value.b.getValue())
             System.out.println("a value is small");
  else
             System.out.println("b value is small");


 }
}
```
output: a value is small

3)else if:

This statement perform a task depending on whether a condition is true or false.

```
                    Syntex: if(condition )
                                Statement
                                        Else if(condition)
                                        Statement
                                Else
                                        Statement
```

program 1:

```
enum Gender {
                MALE, FEMALE, UNKNOWN;


        }

class cont
{
 public static void main( String args[] )
 {
   Gender s=Gender.FEMALE;
  if(s==Gender.MALE)
            System.out.println("both are equal");
  else if (s==Gender.UNKNOWN)
            System.out.println("Both are  equal");
  else
            System.out.println("Both are not equal");


 }
}
```

program 2:

```
 enum Value {
                a(10), b(20), c(30);

                int a1;
                int getValue(){ return a1;}
```

```
            }

class cont
{
 public static void main( String args[] )
 {
   Gender s=Gender.FEMALE;
  switch(s)
  {
  case MALE:System.out.print("Gender is mail");
            break;
  case FEMALE:System.out.print("Gender is femail");
  break;
  case UNKNOWN:System.out.print("Gender is unknown");

  break;
  default:System.out.print("NON of these");
  }
 }
}
```

output: Gender is femail

Java's Iteration Statements: Javaơs iteration statements are for, while and do-while. These statements are used to repeat same set of instructions specified number of times called loops

Types of looping statements are:

1)while

2)do while

3)for

1) **while Loop:** while loop repeats a group of statements as long as condition is true. Once the condition is false, the loop is terminated. In while loop, the condition is tested first; if it is true, then only the statements are executed. while loop is called as entry control loop.

Syntax: while (condition)

```
   {
     statements;
   }
```

Program: To find Sum of given number:

```
 enum Value {
                    NUM(10);

                    int a1;
                    int getValue(){ return a1;}
                    Value(int value)
                    {
                              this.a1=value;
                    }
          }

class Enu
{
 public static void main( String args[] )
 {
  int n=Value.NUM.getValue();
  int sum=0,i=0;
  while( i<n)
  {
            sum+=i;
            i++;
  }
  System.out.println("sum of given number is="+sum);
}
}
```

output: sum of given number is=45

2)do while Loop: doé while loop repeats a group of statements as long as condition is true. In do...while loop, the statements are executed first and then the condition is tested.

doé while loop is also called as exit control loop.

Syntax: do

```
   {
     statements;
   } while (condition);
```

Program: To find sum of given number:

```
 enum Value {
                    NUM(10);

                    int a1;
                    int getValue(){ return a1;}
                    Value(int value)
```

```
                {
                        this.a1=value;
                }
        }

class Enu
{
 public static void main( String args[] )
 {
  int n=Value.NUM.getValue();
  int sum=0,i=0;
  do
  {
          sum+=i;
          i++;
  }while( i<n);
  System.out.println("sum of given number is="+sum);
}
}
```

output: sum of given number is=45

3.for Loop: The for loop is also same as doé while or while loop, but it is more compact syntactically. The for loop executes a group of statements as long as a condition is true.

Syntax: for (expression1; expression2; expression3)

```
   {    statements;
   }
```

Here, expression1 is used to initialize the variables, expression2 is used for condition checking and expression3 is used for increment or decrement variable value.

program : To find the sum of given number.

```
 enum Value {
                NUM(10);

                int a1;
                int getValue(){ return a1;}
                Value(int value)
                {
                        this.a1=value;
                }
        }

class Enu
```

```
{
 public static void main( String args[] )
 {
  int n=Value.NUM.getValue();
  int sum=0;
  for(int i=0;i<n;i++)
  {
             sum+=i;
  }
  System.out.println("sum of given number is="+sum);
}
}
```

output: sum of given number is=45

2. **Java Enumerations Are Class Types**

- Enumerations in Java can have methods, members and constructors just as any other class can have.
- Each enumeration constant is an object of its enumeration type.
- Thus, when you define a constructor for an **enum**, the constructor is called when each enumeration constant is created.
- Also, each enumeration constant has its own copy of any instance variables defined by the enumeration.
- the enum constants have initial value that starts from 0, 1, 2, 3 and so on. But we can initialize the specific value(default value) to the enum constants by defining fields and constructors.

Syntax:

```
enum variableName
{
 ITEM1(1), ITEM2(20), ITEM3(30);
data-typevariableName;
 data-type methodName()
{ statement;}
enumName (parameter-list) {
            statements;
}

}
```

Progarm:

```
enum Value {
                 A(10), B(20), C(30);

                 int a;
                 int getValue(){ return a;}
```

```
                    }

    }}
```

```
output:
January:31
February:28
March:31
April:30
May:31
June:30
July:31
August:31
September:30
October:31
November:30
December:31
```

2.ValueOf():

- **method returns the enumeration constant whose value corresponds to the string** passed in *str.*
- method takes a single parameter of the constant name to retrieve and returns the constant from the enumeration, if it exists.

Syntax: enumerationVariable = enumerationName.valueOf("EnumerationValueInList");

Example

```
        WeekDays wd = WeekDays.valueOf("MONDAY");
        System.out.println(wd);
```

program:

```
enum Days {
        monday,tuesday;


    }

class cont
{
 public static void main( String args[] )
 {
  Days d=Days.valueOf("monday");
  System.out.println("day selected is:"+d);
 }
}
```

```
output: day selected is:Monday
```

Note: An enumeration cannot inherit another class and an enum cannot be a superclass for other class.

4.Enumerations Inherit Enum:

All enumerations in Java inherit the Enum class, java.lang.Enum, which provide a set of methods for all enumerations. The four mentioned here are ordinal( ) and compareTo( ),equals() and toString().

1.ordinal():

- Returns the value of the constant's position in the list (the first constant has a position of zero).
- The ordinal value provides the order of the constant in the enumeration, starting with 0

Example

```
WeekDays wd = WeekDays.MONDAY;
System.out.println(wd.ordinal());
```

program: To find index of Enum List.

```
enum Days {
                mon,tue,wed;
                }
class cont
{
 public static void main( String args[] )
 {
          Days wd = Days.mon;
          System.out.println("Index of list:"+wd.ordinal());
 }
}
```

output: Index of list:0

Program 2: Using foreach loop

```
enum Days {
                mon,tue,wed;
                }
class cont
{
 public static void main( String args[] )
 {
          Days wd[] = Days.values();
          for(Days w:wd)
```

3.equals():

- method returns true if the specified object is equal to this enum constant.

  public final boolean equals(Object other)

- **where other** ī This is the object to be compared for equality with this object.
- This method returns true if the specified object is equal to this enum constant
- Example:

```
enum Days {
          mon,tue,wed;
          }
          class cont
          {
           public static void main( String args[] )
           {
                    Days d1,d2,d3;
                    d1=Days.mon;
                    d2=Days.tue;
                    d3=Days.wed;
          System.out.println(d1.equals(d2));
          System.out.println(d2.equals(d3));
          System.out.println(d2.equals(d2));
          }
          }
          Output:
          false
          false
          true
```

4.toString():

method returns the name of this enum constant, as contained in the declaration.

public String toString()

This method returns the name of this enum constant.

Example:

```
enum Days {
                mon,tue,wed;
                }
                class cont
                {
                 public static void main( String args[] )
                 {
                          Days d1,d2,d3;
                          d1=Days.mon;
                          d2=Days.tue;
                          d3=Days.wed;
                System.out.println(d1.toString());
                System.out.println(d2. toString());
                System.out.println(d3. toString());
                }
```

}

Output: mon tue wed

4.Type wrappers:

- Java uses primitive types (also called simple types), such as int or double, to hold the basic data types supported by the language.
- Instead of primitive types if objects are used everywhere for even simple calculations then performance overhead is the problem.
- So to avoid this java had used primitive types.
- So primitive types do not inherit Object class
- But there are times when you will need an object representation for primitives like int and char.
- Example, you canꞌt pass a primitive type by reference to a method.
- Many of the standard data structures implemented by Java operate on objects, which mean that you canꞌt use these data structures to store primitive types.
- To handle these (and other) situations, Java provides *type wrappers, which are classes* that encapsulate a primitive type within an object.

The type wrappers are :

Double, Float, Long, Integer, Short, Byte,
Character, and Boolean.

Character:

- Character is a wrapper around a char.
- The constructor for Character is Character(char ch) here ch is a character variable whose values will be wrapped to character object by the wrapper class
- To obtain the char value contained in a Character object, call charValue( ), shown here:

  char charValue( )

  It returns the encapsulated character.

Boolean

- **Boolean** is a wrapper around **boolean** values. It defines these constructors:

  Boolean(boolean *boolValue*)
  Boolean(String *boolString*)

- In the first version, *boolValue* must be either **true** or **false**. In the second version, if *boolString* contains the string "true" (in uppercase or lowercase), then the new **Boolean** object will be true. Otherwise, it will be false.
- To obtain a **boolean** value from a **Boolean** object, use **booleanValue( )**, shown here:

boolean booleanValue( )

- It returns the **boolean** equivalent of the invoking object.

The Numeric Type Wrappers

- The most commonly used type wrappers are those that represent numeric values. These are **Byte**, **Short**, **Integer**, **Long**, **Float**, and **Double**. All of the numeric type wrappers inherit the abstract class **Number**.
- **Number** declares methods that return the value of an object in each of the different number formats. These methods are shown here:

1. byte byteValue( )
2. double doubleValue( )
3. float floatValue( )
4. int intValue( )
5. long longValue( )
6. short shortValue( )

doubleValue( ): returns the value of an object as a **double**
floatValue( ): returns the value as a **float**, and so on.

- All of the numeric type wrappers define constructors that allow an object to be constructed from a given value, or a string representation of that value. For example, here are the constructors defined for **Integer**:

Integer(int *num*) Integer(String *str*)

- If *str* does not contain a valid numeric value, then a **NumberFormatException** is thrown. All of the type wrappers override **toString( )**. It returns the human-readable form of the value contained within the wrapper. This allows you to output the value by passing a type wrapper object to **println( )**, for example, without having to convert it into its primitive type.

Program : All wrapper class

```
class Wrap {
public static void main(String args[]) {

Character c=new Character('@'); // character type
char c1=c.charValue();
System.out.println("Character wrapper class"+c1);

Boolean b=new Boolean(true);
boolean b1=b.booleanValue();
System.out.println("Boolean wrapper class"+b1);
Integer i1 = new Integer(100);  // integre type
int i = i1.intValue();
System.out.println("Integer wrapper class"+i); // displays 100 100

Float f1 = new Float(12.5);  // Float type
float f = f1.floatValue();
System.out.println("Float wrapper class"+f);

}

   }

   output:

Character wrapper class@
Boolean wrapper classtrue
Integer wrapper class100
Float wrapper class12.5
```

Autoboxing

- **Autoboxing** is the process by which a primitive type is automatically encapsulated (boxed) into its equivalent type wrapper whenever an object of that type is needed. There is no need to explicitly construct an object.
- For example, converting int to Integer class. The Java compiler applies autoboxing when a primitive value is:
    - Passed as a parameter to a method that **expects an object** of the corresponding wrapper class.
    - Assigned to a variable of the corresponding **wrapper class**.
- **Auto-unboxing** is the process by which the value of a boxed object is automatically extracted(unboxed) from a type wrapper when its value is needed. There is no need to call a method such as **intValue( ) or doubleValue( ).**
- For example conversion of Integer to int. The Java compiler applies unboxing when an object of a wrapper class is:
    - Passed as a parameter to a method that **expects a value** of the corresponding primitive type.
    - Assigned to a variable of the corresponding **primitive type**.

Uses of Autoboxing and Unboxing

- Useful in removing the difficulty of manually boxing and unboxing values in several algorithms.
- it is very important to generics, which operates only on objects.
- It also helps prevent errors.
- autoboxing makes working with the Collections Framework
- here is the modern way to construct an **Integer object that**

```
//    Autobox/unbox a char.
    Character ch = 'x'; // box a char
    char ch2 = ch; // unbox a char
    System.out.println("ch2 is " + ch2);

}
}
output:
b is true
ch2 is x
```

Autoboxing/Unboxing Helps Prevent Errors:

- Autoboxing always creates the proper object and auto unboxing always produce the proper value.
- There is no wayfor the process to produce the wrong type of object or value.

Program:

```
class auto {
public static void main(String args[]) {

Integer iOb = 1000; // autobox the value 1000
int i = iOb.byteValue(); // manually unbox as byte !!!
System.out.println("unbox value:"+i); // does not display 1000 !
}
}
```

```
                        output:
                        unbox value:-24
```

▪ This program displays not the expected value of 1000, but ï 24! The reason is that the value inside **iOb** is manually unboxed by calling byteValue( ), which causes the truncation of the value stored in iOb, which is 1,000.

▪ This results in the garbage value of ï 24 being assigned to i.

▪ Auto-unboxing prevents this type of error because the value in iOb will always autounbox into a value compatible with int.

A Word of Warning:

Because of autoboxing and auto-unboxing, some might be tempted to use objects such as **Integer** or **Double** exclusively, abandoning primitives altogether.

```
Double a, b, c;
a = 10.0;
b = 4.0;
c = Math.sqrt(a*a + b*b);
System.out.println("Hypotenuse is " + c);
```

Annotation :

- Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
- Annotations in java are used to provide additional information, so it is an alternative option for XML and java marker interfaces

What's the use of Annotations?

1) Instructions to the compiler: There are three built-in annotations available in Java (@Deprecated, @Override & @SuppressWarnings) that can be used for giving certain instructions to the compiler. For example the @override annotation is used for instructing compiler that the annotated method is overriding the method.

2) Compile-time instructors: Annotations can provide compile-time instructions to the compiler that can be further used by sofware build tools for generating code, XML files etc.

3) Runtime instructions: We can define annotations to be available at runtime which we can access using java reflection and can be used to give instructions to the program at runtime.

Annotation basics

- An annotation is created through a mechanism based on the **interface.**
- A Java annotation in its shortest form looks like this:

  Systex: @interface MyAnno

- The @ that precedes the keyword **interface**. This tells the compiler that an annotation type is being declared The name following the @interface character is the name of the annotation.
- All annotations consist solely of method declarations.
- However, you donôt provide bodies for these methods. Instead, Java implements these methods. Moreover, the methods act much like fields.
- Annotations can be applied to the classes, interfaces, methods and fields. For example the below annotation is being applied to the method.

```
@Override
void myMethod() {
   //Do something
}
```

- An annotation cannot include an **extends** clause. However, all annotation types automatically extend the **Annotation** interface. Thus, **Annotation** is a super-interface of

Example:

```
@Retention(RetentionPolicy.RUNTIME)
@interface MyAnno {
 String str(); int val();
 }
```

Obtaining Annotations at Run Time by Use of Reflection:

- Reflection is the feature that enables information about a class to be obtained at run time. The reflection API is contained in the **java.lang.reflect** package.
- The first step to using reflection is to obtain a **Class** object that represents the class whose annotations you want to obtain. **Class** is one of Javaȏs built-in classes and is defined in **java.lang**. There are various ways to obtain a **Class** object. One of the easiest is to call **getClass( )**, which is a method defined by **Object**. Its general form is shown here:

final Class<?> getClass( )

- It returns the Class object that represents the invoking object.
- After you have obtained a **Class** object, you can use its methods to obtain information about the various items declared by the class, including its annotations **Class** supplies (among others) the **getMethod( )**, **getField( )**, and **getConstructor( )** methods, which obtain information about a method, field, and constructor, respectively. These methods return objects of type **Method**, **Field**, and **Constructor**.

Method getMethod(String *methName*, Class<?> ... *paramTypes*)

- From a **Class**, **Method**, **Field**, or **Constructor** object, you can obtain a specific annotation associated with that object by calling **getAnnotation( )**.

<A extends Annotation> getAnnotation(Class<A> *annoType*)

Program:

```
import java.lang.annotation.*;

import java.lang.reflect.*;
  @Retention(RetentionPolicy.RUNTIME) @interface MyAnno {
```

```
String str(); int val();

}
class annu {

@MyAnno(str = "This is Retention method", val = 100)

public static void show() {
          annu ob = new annu();
    try {
Class<?> c = ob.getClass();
Method m = c.getMethod("show");

    MyAnno anno = m.getAnnotation(MyAnno.class);
    System.out.println(anno.str() + " " + anno.val());
} catch (NoSuchMethodException exc) {

System.out.println("Method Not Found.");
}
}
public static void main(String args[]) { show();
}
}
```

OUTPUT:

```
This is Retention method  100
```

Program 2: Passing an argument to the method.

```
import java.lang.annotation.*;
import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME)

@interface MyAnno {
String str(); int val();
}
class mymethod {

 @MyAnno(str = "Two Parameters", val = 19)

public static void myMeth(String str, int i)
{
mymethod ob = new mymethod();

try {
```

```
Class<?>    c = ob.getClass();

Method m   = c.getMethod("myMeth",       String.class, int.class);

MyAnno anno = m.getAnnotation(MyAnno.class);

System.out.println(anno.str() + " " + anno.val());
}
catch (NoSuchMethodException exc) {
System.out.println("Method Not Found.");

}
}
public static void main(String args[]) { myMeth("test", 10);
}
```

OUTPUT:

```
Two Parameters 19
```

The Annotated Element Interface:

- The methods **getAnnotation( )** and **getAnnotations( )** are defined by the **AnnotatedElement** interface, which is defined in **java.lang.reflect**.
- This interface supports reflection for annotations and is implemented by the classes **Method**,**Field**, **Constructor**, **Class**, and **Package**, among others.
- In addition to **getAnnotation( )** and **getAnnotations( )**, **AnnotatedElement** defines several other methods.
    - **getDeclaredAnnotations( )**
    - **isAnnotationPresent( )**
    - **getDeclaredAnnotation( )**,
    - **getAnnotationsByType( )**,
    - **getDeclaredAnnotationsByType( )**.

```
}

}
```

output:

```
@MyAnno(str=This is Retention method and  value  is , val=100)
```

isAnnotationPresent( ):

- Method returns true if an annotation for the specified type is present on this element, else false. This method is designed primarily for convenient access to marker annotations.

Syntax: public boolean isAnnotationPresent(Class<? extends Annotation> annotationClass)

Program:

```
import java.lang.annotation.*;

import java.lang.reflect.*;

@Retention(RetentionPolicy.RUNTIME) @interface MyAnno {
String str(); int val();

}
class annu {

@MyAnno(str = "This is Retention method and  value  is ", val = 100)

public static void main(String args[]) {

Package[] pack = Package.getPackages();


// check if annotation hello exists
for (int i = 0; i < pack.length; i++) {
  System.out.println("" + pack[0].isAnnotationPresent(MyAnno.class));
}

}
```

OUTPUT: Falseé .

getDeclaredAnnotation( ):

Method returns this element's annotation for the specified type if such an annotation is present, else null.This methods throws an exception

- **NullPointerException** ī if the given annotation class is null
- **IllegalMonitorStateException** ī if the current thread is not the owner of the object's monitor.

Syntax: public <A extends Annotation> A getAnnotation(Class<A> annotationClass)

Program:

```
import java.lang.annotation.*;
import java.lang.reflect.*;
  @Retention(RetentionPolicy.RUNTIME) @interface MyAnno {
String str(); int val();
}
class annu {
@MyAnno(str = "This is Retention method and  value  is ", val = 100)

public static void show() {
          annu ob = new annu();

     try {
Class<?> c = annu.class;
Method m = c.getMethod("show");
    MyAnno anno = m.getAnnotation(MyAnno.class);
     System.out.println(anno.str() + " " + anno.val());
} catch (NoSuchMethodException exc) {

System.out.println("Method Not Found.");
}
}
public static void main(String args[]) { show();
}

}
```

output:

```
This is Retention method and  value  is  100
```

getAnnotationsByType( ):

Returns annotations that are *associated* with this element. If there are no annotations *associated* with this element, the return value is an array of length 0.

program:

```
import java.lang.annotation.Repeatable;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

@Retention(RetentionPolicy.RUNTIME)
@interface LogHistory {
  Log[] value();
}
@Repeatable(LogHistory.class)
@interface Log {
  String date();
  String comments();
}

@Log(date = "02/01/2014", comments = "A")
@Log(date = "01/22/2014", comments = "B")

public class HelloWorld {
  public static void main(String[] args) {
    Class<HelloWorld> mainClass = HelloWorld.class;

    Log[] annList = mainClass.getAnnotationsByType(Log.class);
    for (Log log : annList) {
      System.out.println("Date=" + log.date() + ", Comments=" + log.comments());
    }
}
}
```

output:

```
Date=02/01/2014, Comments=A

Date=01/22/2014, Comments=B
```

<u>Using Default Values:</u>

we can give annotation members default values that will be used if no value is specified when the annotation is applied. A default value is specified by adding a **default** clause to a memberôs declaration. It has this general form:

type member( ) default *value* ;

Program:

```
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@interface MyINF {
String str() default "WELCOME";
int val() default 5;

}
class annu {
@MyINF()
public static void myMeth() {
annu ob = new annu();
try {
Class<?> c = ob.getClass();
Method m = c.getMethod("myMeth");
MyINF anno = m.getAnnotation(MyINF.class);
System.out.println(anno.str() + " " + anno.val());
}
catch (NoSuchMethodException exc) {
System.out.println("Method Not Found.");
}
}
public static void main(String args[]) {
myMeth();
}

}
```

```
MyMarker is present
```

Single-Member Annotations:

- A *single-member* annotation contains only one member. It works like a normal annotation except that it allows a shorthand form of specifying the value of the member. When only one member is present, you can simply specify the value for that member when the annotation is appliedð you donôt need to specify the name of the member. However,
- In order to use this shorthand, the name of the member must be **value**. Here is an example that creates and uses a single-member annotation:

Syntax:

```
public @interface Example{

  String showSomething();

   }
```

Program:

```
import java.lang.annotation.*;
import java.lang.reflect.*;
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@interface MyINF {
//String str() ;
        int value();
}
class annu {
@MyINF(100)
public static void myMeth() {
annu ob = new annu();
try {
Class<?> c = ob.getClass();
Method m = c.getMethod("myMeth");
MyINF anno = m.getAnnotation(MyINF.class);
System.out.println(anno.value());
}
catch (NoSuchMethodException exc) {
System.out.println("Method Not Found.");
}
}
public static void main(String args[]) {
myMeth();
}

}
```

output:

```
100
```

The Built-In Annotations:

- Java defines seven built-in annotations out of which three (@Override, @Deprecated, and @SuppressWarnings) are applied to Java code and they are included in java.lang library. These three annotations are called *regular Java annotations*.
- Rest four (@Retention, @Documented, @Target, and @Inherited) are applied to other annotations and they are included in java.lang.annotation library. These annotations are called *meta Java annotations*.

Annotation Name	**Applicable To**	**Use**	**Included in**
Java Annotations Applied to Java code			
@Override	Member Methods	Checks that this method overrides a method from its superclass	java.lang
@Deprecated	All annotable items	Marks item as deprecated	java.lang
@SuppressWarnings	All annotable items except packages and annotations	Suppress warning of given type	java.lang
Java Annotations Applied to Other Annotations			
@Retention	Annotations	Specifies how long this annotation is retained -	java.lang.annotation

		whether in code only, compiled into the class, or available at run time through reflection.	
@Documented	Annotations	Specifies that this annotation should be included in the documentation of annotated items	java.lang.annotation
@Target	Annotations	Specifies the items to which this annotation can be applied	java.lang.annotation
@Inherited	Annotations	Specifies that this annotation, when applied to a class, is automatically inherited by its subclasses.	java.lang.annotation

@Override:

@**Override** is a marker annotation that can be used only on methods. A method annotated with **@Override** must override a method from a superclass. If it doesnǿt, a compile-time error will result. It is used to ensure that a superclass method is actually overridden, and not simply overloaded

Example:

```
public class Animal {

    public void makeSound(){

    }
}

class Cat extends Animal{

    @Override
    public void makeSound(){
        System.out.println("myyyyyaaawwwwww");
    }
}
```

@Deprecated

Use this annotation on methods or classes which you ***need to mark as deprecated***. Any class that will try to use this deprecated class or method, will get a compiler ñ**warning**ñ.

```
@Deprecated public Integer myMethod()

{

   return null;

}
```

@SuppressWarnings

This annotation ***instructs the compiler to suppress the compile time warnings*** specified in the annotation parameters. e.g. to ignore the warnings of unused class attributes and methods use @SuppressWarnings("unused") either for a given attribute or at class level for all the unused attributes and unused methods.

```
@SuppressWarnings("unused")

public class DemoClass

{

    //@SuppressWarnings("unused")

    private String str = null;

      //@SuppressWarnings("unused")

    private String getString(){

      return this.str;

    }

}
```

@Target Java Annotation

- While defining a custom Java annotation we have to specify which element (class, method, field, constructor etc.) this newly defined annotation would be applicable on. The @Target annotation is used for that purpose to set the target elements on which the custom annotation can be applied

- The possible values of elements for @Target annotation. They belong to the enumerated type ElementType.

Element Type	**Annotation Applies To**
ANNOTATION_TYPE	Annotation type declarations
PACKAGE	Packages
TYPE	Classes (including enum) and interfaces (including annotation types)
METHOD	Methods
CONSTRUCTOR	Constructors
FIELD	Fields (including enum constants)
PARAMETER	Method or constructor parameters
LOCAL_VARIABLE	Local variables

example:

```
import java.lang.annotation.ElementType;
import java.lang.annotation.Target;

@Target({ElementType.METHOD})
public @interface MyCustomAnnotation {

}
public class MyClass {
   @MyCustomAnnotation
   public void myMethod()
   {
       //Doing something
   }
}
```

@Retention Java Annotation

- As name suggets, @Retention meta annotation specifies till what level an annotation will be retained. To decide the scope of the custom annotation we have to specify one of the three values (SOURCE, CLASS, or RUNTIME) of RetentionPolicy. The default is RetentionPolicy.CLASS.
  - RetentionPolicy.SOURCE specifies the scope of custom annotation to the compile time. Annotations having retention policy RetentionPolicy.SOURCE are not included in bytecode.
  - Annotations those are carrying RetentionPolicy.CLASS policy of retention are included in .class files, but the virtual machine need not to load them.
  - Annotations having RetentionPolicy.RUNTIME policy are included in class files and loaded by the virtual machine. Java annotations that are given RUNTIME retention policy can be accessed at run time through the reflection API.

example:

```
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;

//@Retention(RetentionPolicy.CLASS)
@Retention(RetentionPolicy.RUNTIME)
//@Retention(RetentionPolicy.SOURCE)
public @interface MyCustomAnnotation
{
   //some code
}
```

@Documented Java Annotation

The @Documented meta annotation hints Javadoc tool to include this annotation in the documentation wherever it is used. Documented Java annotations should be treated just like other modifiers, such as protected or static, for documentation purposes. The use of other annotations is not included in the documentation.

Example:

```
java.lang.annotation.Documented
@Documented
public @interface MyCustomAnnotation {
  //Annotation body
}
@MyCustomAnnotation
public class MyClass {
    //Class body
}
```

@Inherited Java Annotation

The @Inherited annotation can be applied only to annotations for classes. When a superclass is annotated with an @Inherited Java annotation then all of its subclasses automatically have the same annotation.

Example:

```
java.lang.annotation.Inherited

@Inherited
public @interface MyCustomAnnotation {

}
```

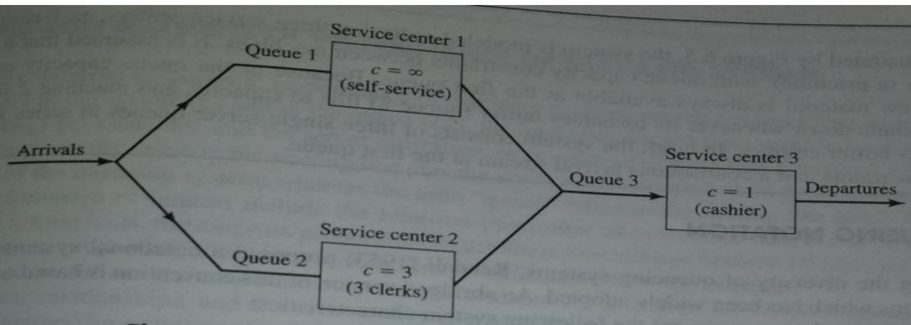


Figure 6.3 Discount warehouse with three service centers.

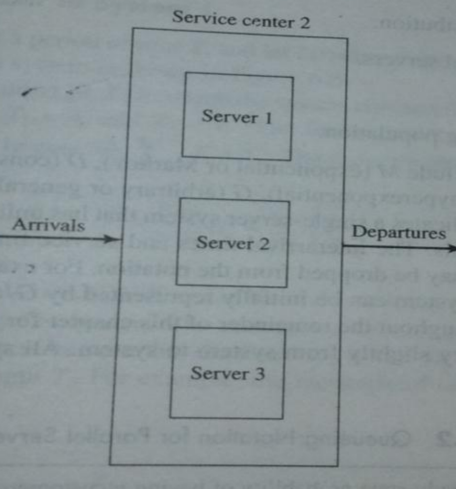


Figure 6.4 Service center 2, with $c = 3$ parallel servers.

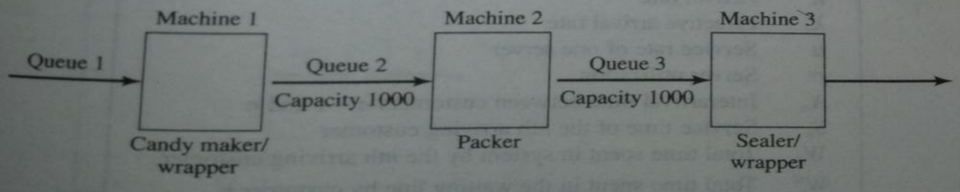


Figure 6.5 Candy-production line.

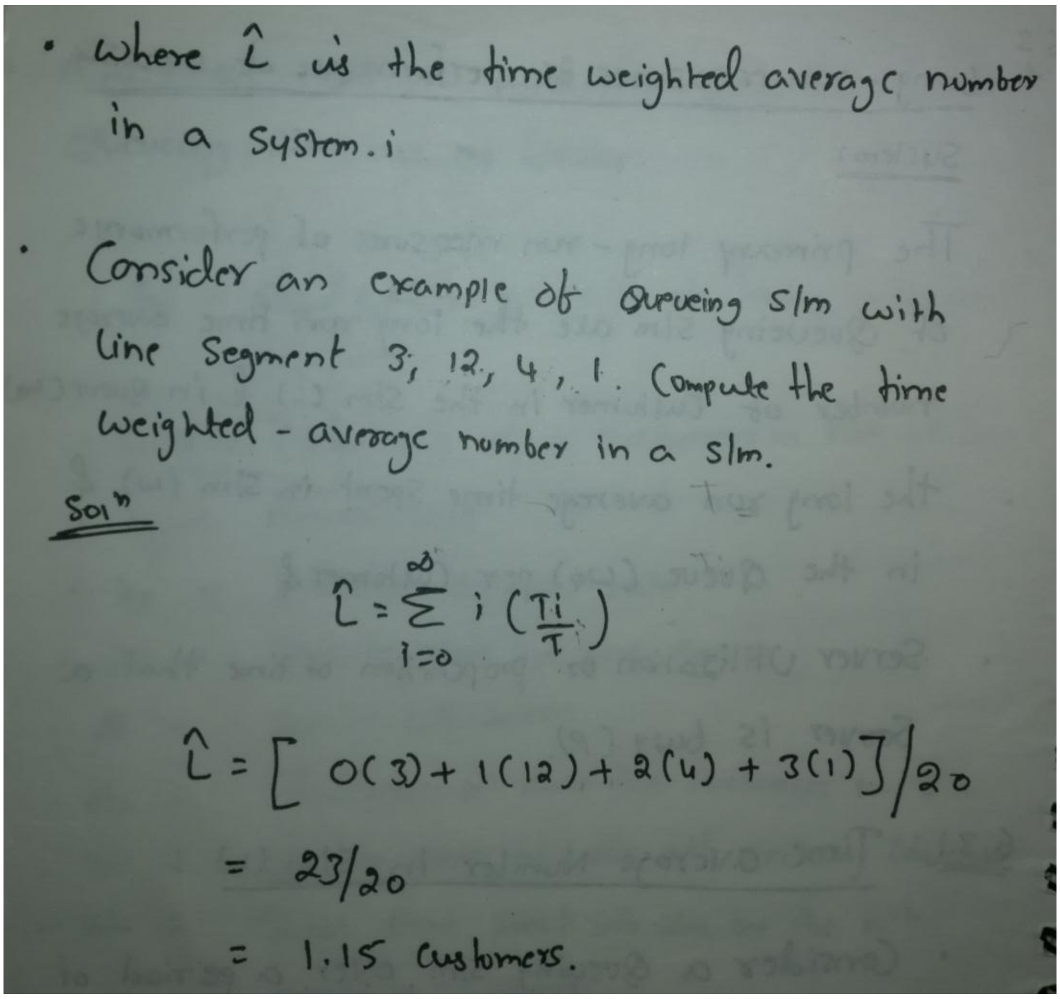
4.2 <u>Queueing Notation(Kendal's Notation)</u>

- Kendal's proposal a notational s/m for parallel server s/m which has been widely adopted.
- An a bridge version of this convention is based on format A|B|C|N|K
- These letters represent the following s/m characteristics:

A-Represents the InterArrival Time distribution
B-Represents the service time distribution
C-Represents the number of parallel servers
N-Represents the s/m capacity
K-Represents the size of the calling populations

Common symbols for A & B include M(exponential or Markov), D(constant or deterministic), E_k (Erlang of order k), PH (phase-type), H(hyperexponential), G(arbitrary or general), & GI(general independent).

- For eg, M|M|1|∞|∞ indicates a single server s/m that has unlimited queue capacity & an infinite population of potential arrivals
- The interarrival tmes & service times are exponentially distributed when N & K are infinite, they may be dropped from the notation.
- For eg, , M|M|1|∞|∞ is often short ended to M|M|1. The tire-curing s/m can be initially represented by G|G|1|5|5.



• where $\hat{L}$ is the time weighted average number in a System. i

• Consider an example of Queueing s/m with line Segment 3, 12, 4, 1. Compute the time weighted - average number in a s/m.

Soln

$$\hat{L} = \sum_{i=0}^{\infty} i \left(\frac{T_i}{T}\right)$$

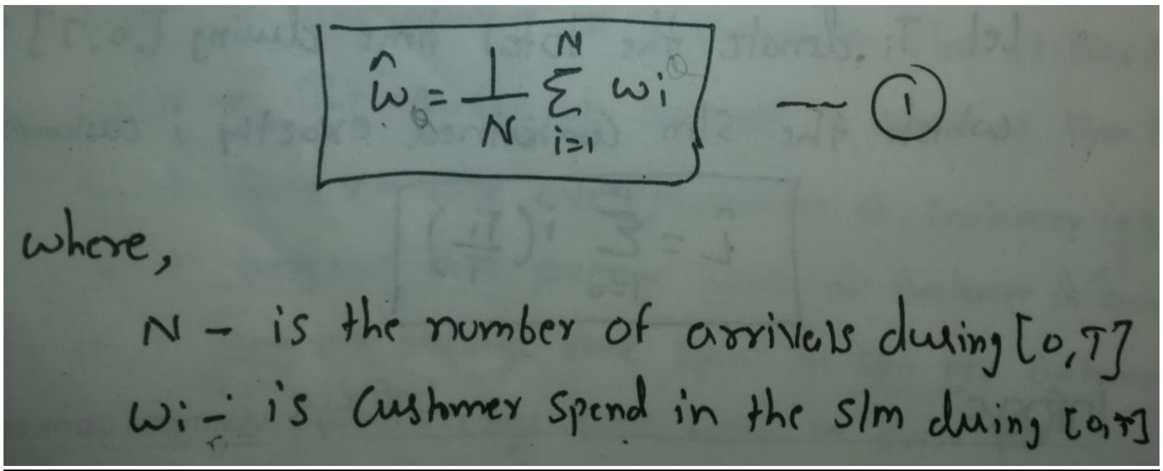
$$\hat{L} = \left[0(3) + 1(12) + 2(4) + 3(1) \right] / 20$$

$$= 23/20$$

$$= 1.15 \text{ customers.}$$

4.3.2 Average Time spent in s/m per customer (w):

- Average s/m time is given as:



$$\boxed{\hat{w} = \frac{1}{N} \sum_{i=1}^{N} w_i} \quad \text{--- } ①$$

where,

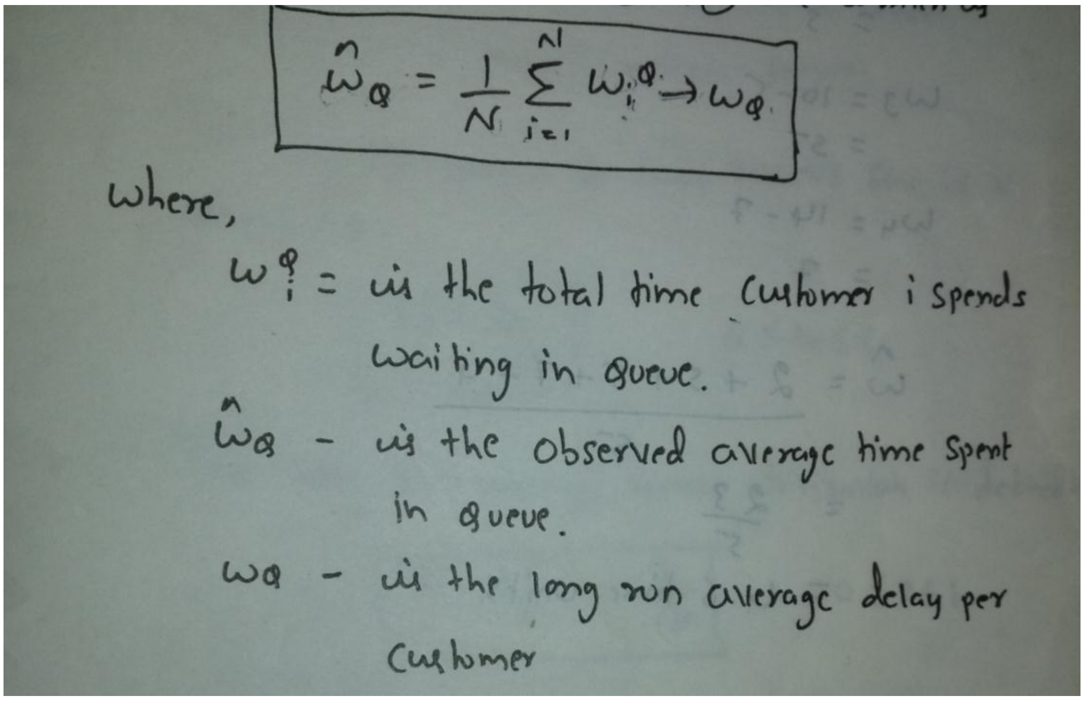
N - is the number of arrivals during [0,T]

w_i - is Customer Spend in the s/m during [0,T]

- For stable s/m N-> ∞

With probability 1, where w is called the long-run average s/m time.

- Considering the equation 1 & 2 are written as,



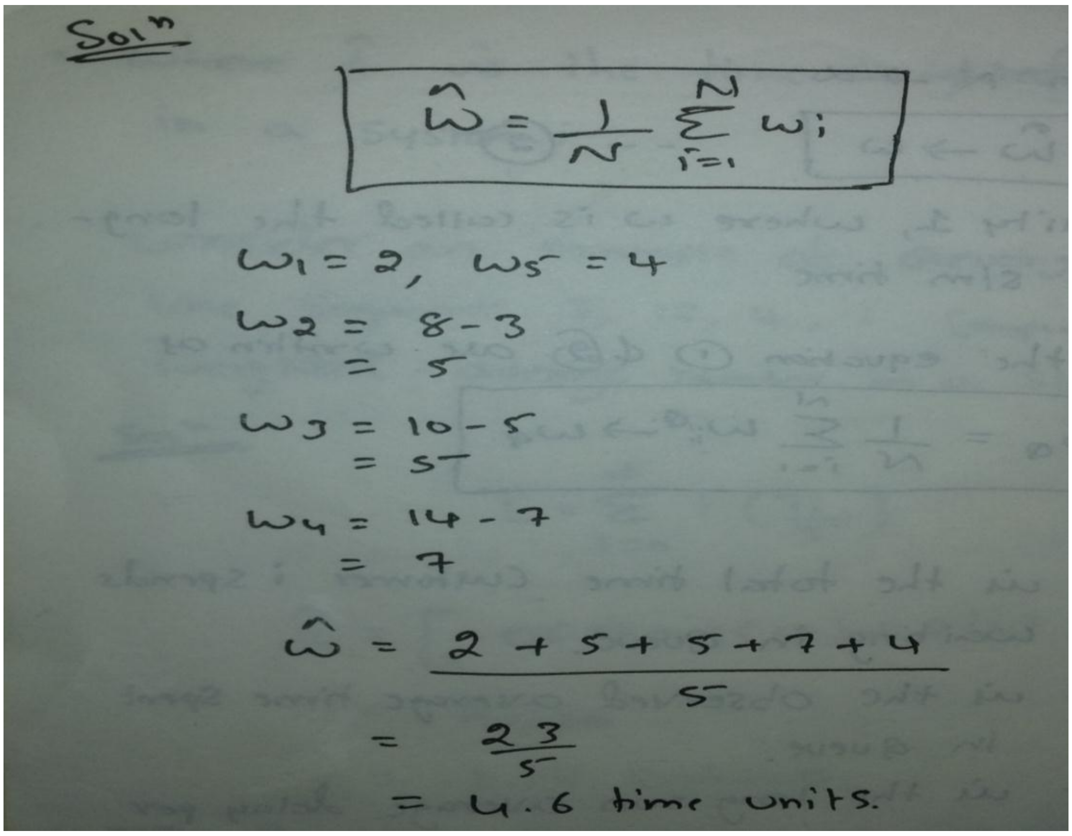
$$\boxed{\hat{\omega}_Q = \frac{1}{N}\sum_{i=1}^{N} \omega_i^Q \rightarrow \omega_Q}$$

where,

w_i^Q = is the total time customer i spends waiting in queue.

$\hat{\omega}_Q$ - is the observed average time spent in queue.

ω_Q - is the long run average delay per customer



Soln

$$\boxed{\hat{\omega} = \frac{1}{N}\sum_{i=1}^{N} \omega_i}$$

$\omega_1 = 2, \quad \omega_5 = 4$

$\omega_2 = 8 - 3$
$= 5$

$\omega_3 = 10 - 5$
$= 5$

$\omega_4 = 14 - 7$
$= 7$

$$\hat{\omega} = \frac{2 + 5 + 5 + 7 + 4}{5}$$
$$= \frac{23}{5}$$
$= 4.6$ time units.

4.3.3 Server utilization:

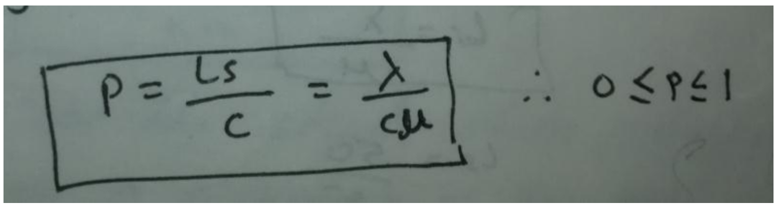
- Server utilization is defined as the population of time server is busy
- Server utilization is denoted by ꞵ is defined over a specified time interval[01]
- Long run server utilization is denoted by p

$\boxed{Ƥ \text{ -> } P}$ as T -> ∞

- **Server utilization in G|G|C|∞|∞ queues**
  - Consider a queuing s/m with c identical servers in parallel
  - If arriving customer finds more than one server idle the customer choose a server without favoring any particular server.
  - The average number of busy servers say Ls is given by,

$\boxed{Ls = \lambda / \mu}$ 0<= Ls <= C

- The long run average server utilization is defined by



$$\boxed{P = \frac{Ls}{c} = \frac{\lambda}{c\mu}} \quad \therefore\ 0 \le P \le 1$$

- The utilization P can be interpreted as the proportion of time an arbitrary server is busy in the long run

4.4 STEADY-STATE BEHAVIOUR OF INFINITE-POPULATION MARKOVIAN MODLES

- For the infinite population models, the arrivals are assumed to follow a poisson process with rate λ arrivals per time unit
- The interarrival times are assumed to be exponentially distributed with mean 1/λ
- Service times may be exponentially distributed(M) or arbitrary(G)
- The queue discipline will be FIFO because of the exponential distributed assumptions on the arrival process, these model are called "MARKOVIAN MODEL".
- The steady-state parameter L, the time average number of customers in the s/m can be computed as

$$\boxed{= \sum_{=0}^{\infty}}$$

Where Pn are the steady state probability of finding n customers in the s/m

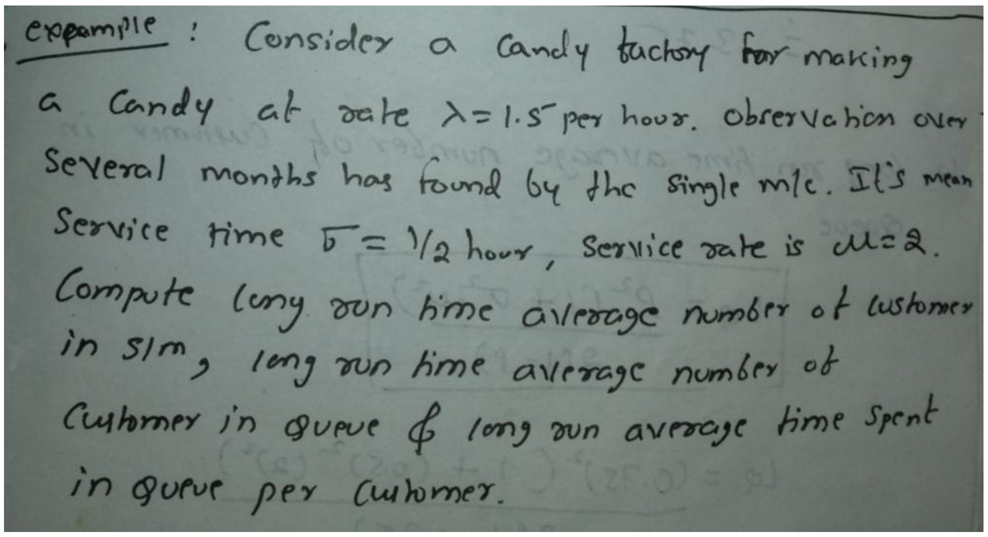
- Other steady state parameters can be computed readily from little equation to whole system & to queue alone

$$\boxed{\begin{aligned} w &= L/\lambda \\ w_Q &= w - (1/\mu) \\ L_Q &= \lambda w_Q \end{aligned}}$$

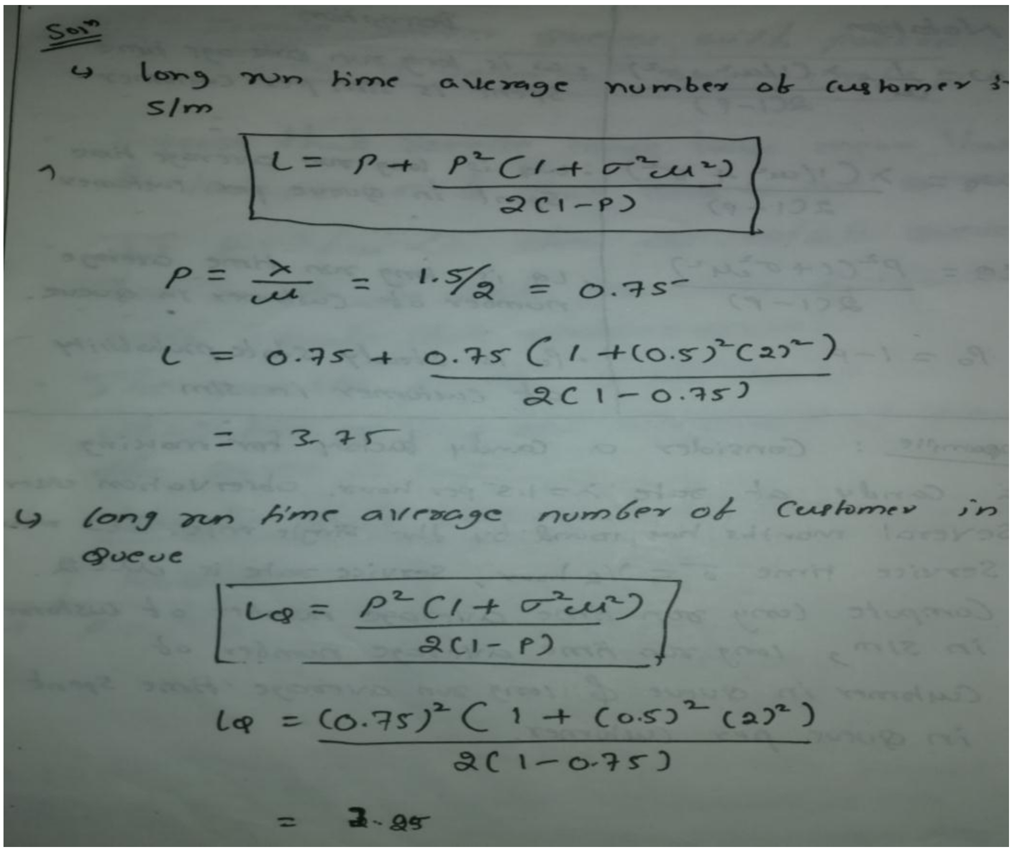
Where λ is the arrival rate & μ is the service rate per server

4.4.1 SINGLE-SERVER QUEUE WITH POISSON ARRIVALS & UNLIMITED CAPACITY: M|G|1

- Suppose that service times have mean $1/\mu$ & variance σ^2 & that there is one server
- If $P = \lambda / \mu < 1$, then the M|G|1 queue has a steady state probability distribution with steady state characteristics
- The quantity $P = \lambda / \mu$ is the server utilization or lon run proportion of time the server is busy
- Steady state parameters of the M|G|1 are:



example : Consider a Candy factory for making a Candy at rate $\lambda = 1.5$ per hour. observation over Several months has found by the single m/c. It's mean Service time $\sigma = 1/2$ hour, Service rate is $\mu = 2$. Compute long run time average number of customer in s/m, long run time average number of Customer in queue & long run average time Spent in queue per Customer.



Soln

↳ long run time average number of customer in s/m

$$\boxed{L = \rho + \frac{\rho^2 (1 + \sigma^2 \mu^2)}{2(1-\rho)}}$$

$$\rho = \frac{\lambda}{\mu} = 1.5/2 = 0.75$$

$$L = 0.75 + \frac{0.75 (1 + (0.5)^2 (2)^2)}{2(1 - 0.75)}$$

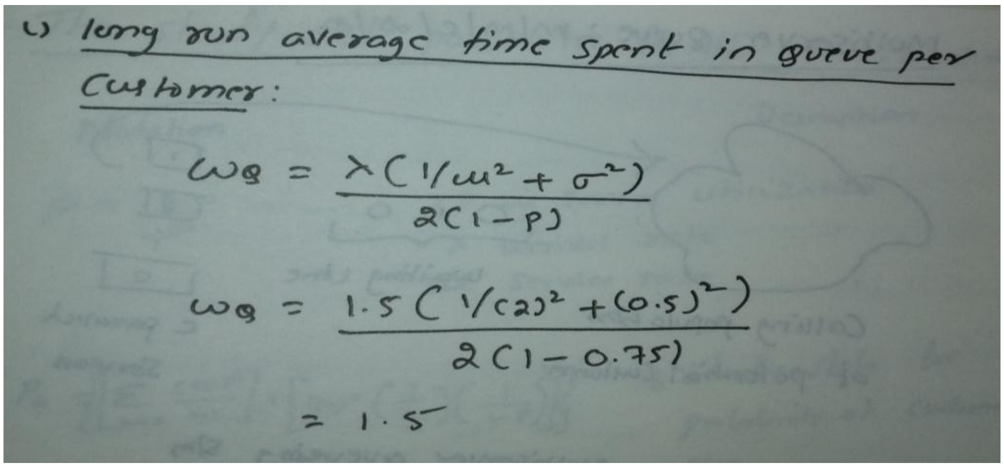
$$= 3.75$$

↳ long run time average number of Customer in Queue

$$\boxed{L_Q = \frac{\rho^2 (1 + \sigma^2 \mu^2)}{2(1-\rho)}}$$

$$L_Q = \frac{(0.75)^2 (1 + (0.5)^2 (2)^2)}{2(1 - 0.75)}$$

$$= 2.25$$

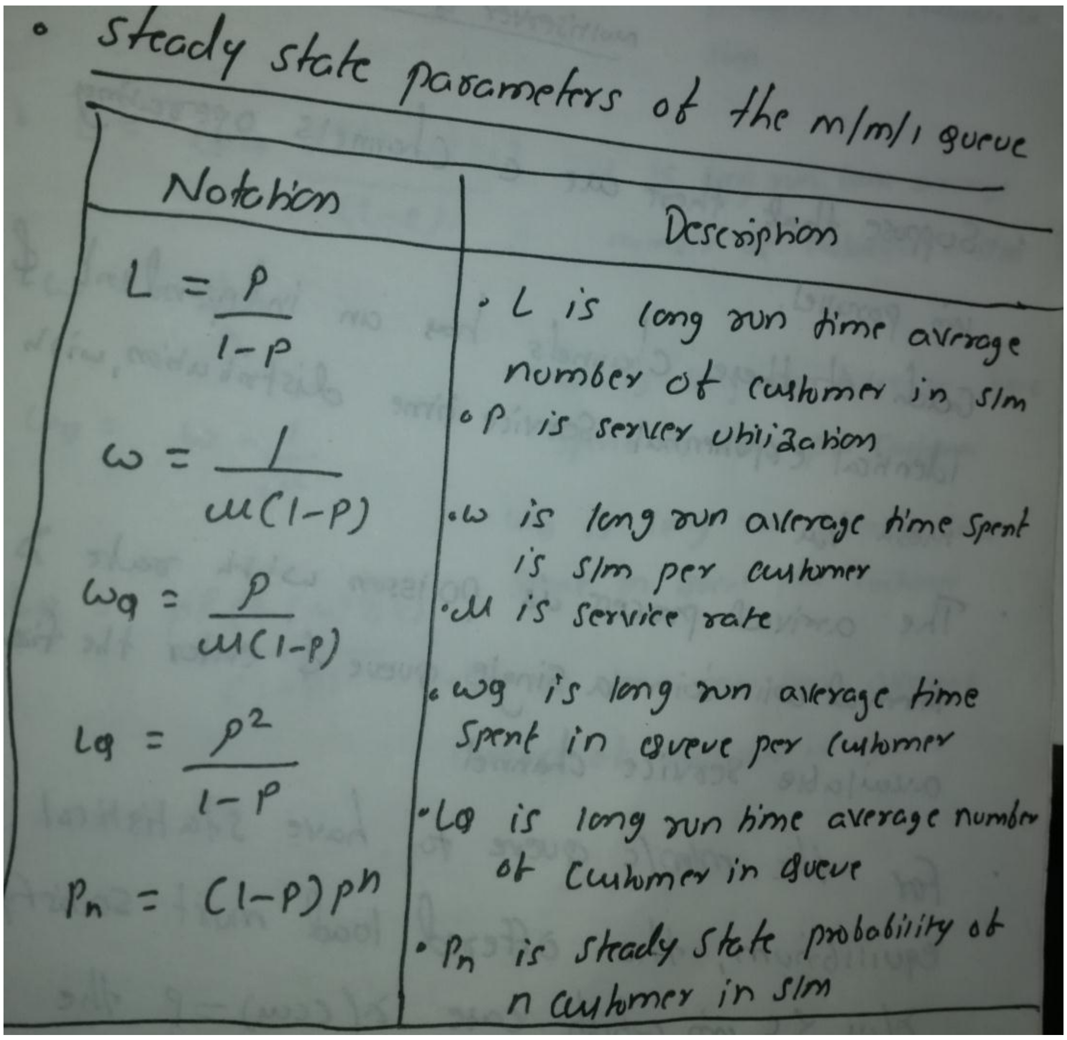


↳ long run average time spent in queue per Customer:

$$w_Q = \frac{\lambda (1/\mu^2 + \sigma^2)}{2(1-\rho)}$$

$$w_Q = \frac{1.5 (1/(2)^2 + (0.5)^2)}{2(1 - 0.75)}$$

$$= 1.5$$



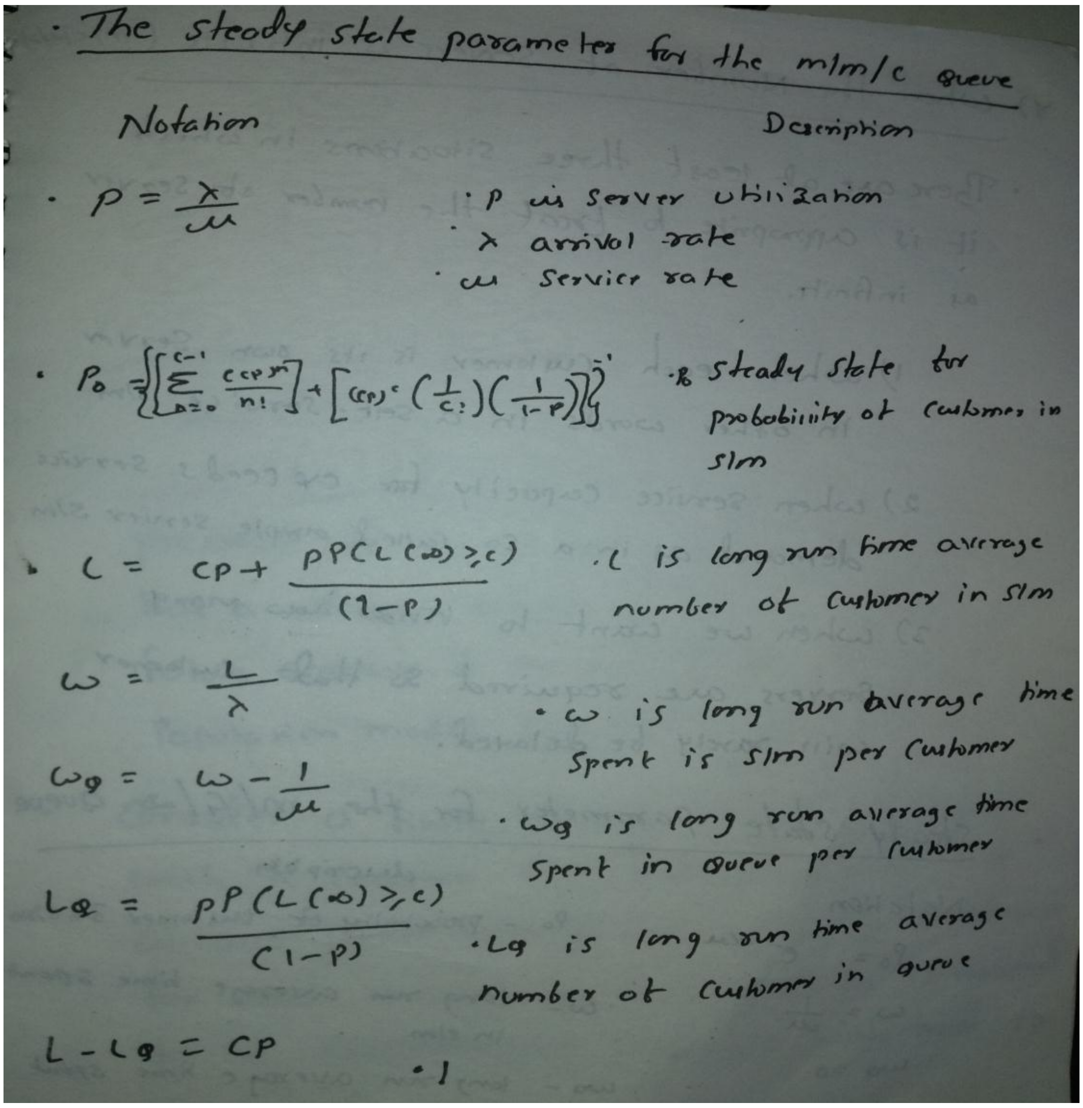
• Steady state parameters of the m/m/1 queue

Notation	Description
$L = \frac{\rho}{1-\rho}$	• L is long run time average number of customer in s/m • ρ is server utilization
$\omega = \frac{1}{\mu(1-\rho)}$	• ω is long run average time spent is s/m per customer • μ is service rate
$\omega_q = \frac{\rho}{\mu(1-\rho)}$	• ω_q is long run average time spent in queue per customer
$L_q = \frac{\rho^2}{1-\rho}$	• L_q is long run time average number of customer in queue
$P_n = (1-\rho)\rho^n$	• P_n is steady state probability of n customer in s/m

4.4 2 MULTISERVER QUEUE: M|M|C|∞|∞

- Suppose that there are c channels operating in parallel
- Each of these channels has an independent & identical exponential service time distribution with mean $1/\mu$
- The arrival process is poisson with rate λ. Arrival will join a single queue & enter the first available service channel

- For the M|M|C queue to have statistical equilibrium the offered load must satisfy λ/μ <c in which case λ/ (cμ) = P the server utilization.



· The steady state parameters for the m/m/c queue

Notation — Description

- $\rho = \frac{\lambda}{\mu}$ — · ρ is server utilization · λ arrival rate · μ service rate
- $P_0 = \left\{\left[\sum_{n=0}^{c-1} \frac{(c\rho)^n}{n!}\right] + \left[(c\rho)^c \left(\frac{1}{c!}\right)\left(\frac{1}{1-\rho}\right)\right]\right\}^{-1}$ — · P₀ steady state for probability of customer in s/m
- $L = c\rho + \frac{\rho P(L(\infty) \geq c)}{(1-\rho)}$ — · L is long run time average number of customer in s/m

$w = \frac{L}{\lambda}$ — · w is long run average time spent is s/m per customer

$w_q = w - \frac{1}{\mu}$ — · wq is long run average time spent in queue per customer

$L_q = \frac{\rho P(L(\infty) \geq c)}{(1-\rho)}$ — · Lq is long run time average number of customer in queue

$L - L_q = c\rho$ · 1

WHEN THE NUMBER OF SERVERS IS INFINITE (M|c|∞|∞)

- There are at least three situations in which it is appropriate to treat the number of server as infinite
  1. When each customer is its own server in other words in a self service s/m
  2. When service capacity far exceeds service demand as in a so called ample server s/m
  3. When wee want to know how many servers are required so that customer will rarely be delayed.

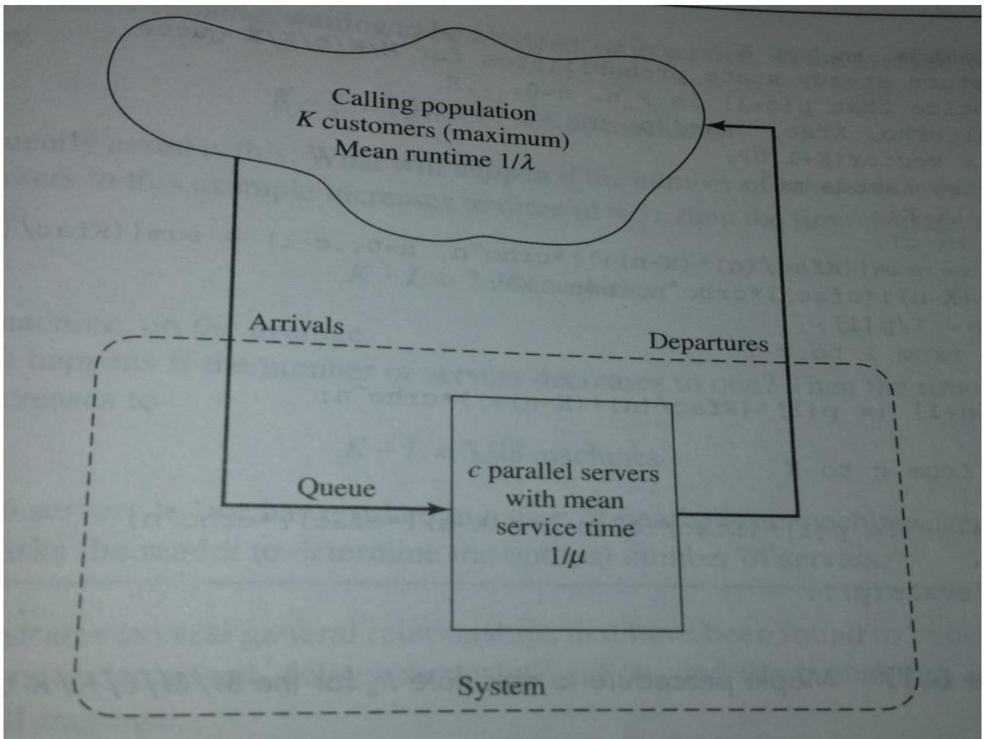


Figure 6.16 Finite-population queueing model.

The effective arrival rate λ_e has several valid interpretations:

Λ_e = long-run effective arrival rate of customers to queue
= long-run effective arrival rate of customers entering service
= long-run rate at which customers exit from service
= long-run rate at which customers enter the calling population
=long-run rate at which customers exit from the calling population.

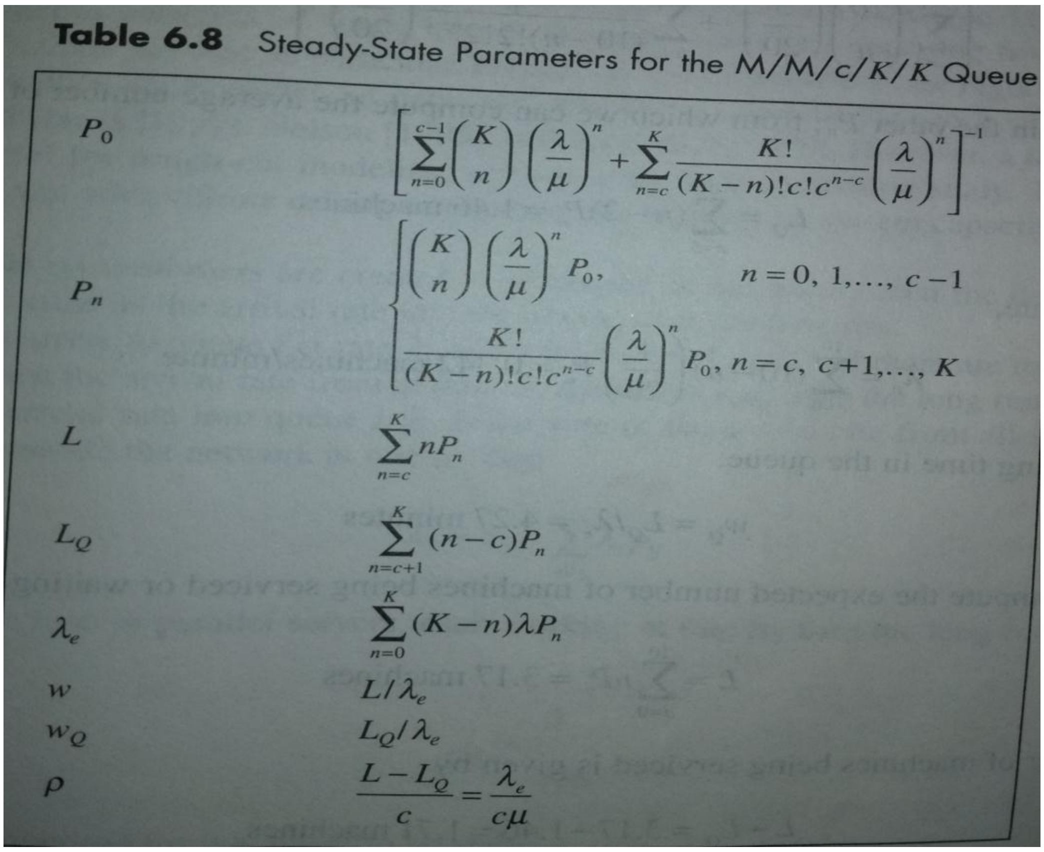


Table 6.8 Steady-State Parameters for the M/M/c/K/K Queue

P_0	$\left[\sum_{n=0}^{c-1}\binom{K}{n}\left(\frac{\lambda}{\mu}\right)^n+\sum_{n=c}^{K}\frac{K!}{(K-n)!c!c^{n-c}}\left(\frac{\lambda}{\mu}\right)^n\right]^{-1}$
P_n	$\begin{cases}\binom{K}{n}\left(\frac{\lambda}{\mu}\right)^n P_0, & n=0,1,\ldots,c-1\\ \frac{K!}{(K-n)!c!c^{n-c}}\left(\frac{\lambda}{\mu}\right)^n P_0, & n=c,c+1,\ldots,K\end{cases}$
L	$\sum_{n=c}^{K} nP_n$
L_Q	$\sum_{n=c+1}^{K}(n-c)P_n$
λ_e	$\sum_{n=0}^{K}(K-n)\lambda P_n$
w	L/λ_e
w_Q	L_Q/λ_e
ρ	$\frac{L-L_Q}{c}=\frac{\lambda_e}{c\mu}$

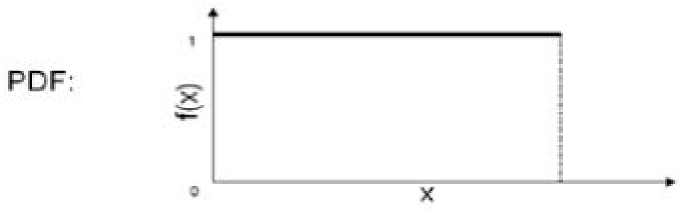
UNIT 5:Random number generation And Variation Generation

RANDOM-NUMBER GENERATION Random numbers are a necessary basic ingredient in the simulation of almost all discrete systems. Most computer languages have a subroutine, object, or function that will generate a random number. Similarly simulation languages generate random numbers that are used to generate event times and other random variables.

5.1 Properties of Random Numbers A sequence of random numbers, R1, R2... must have two important statistical properties, uniformity and independence. Each random number *Ri,* is an independent sample drawn from a continuous uniform distribution between zero and 1.
That is, the pdf is given by

$$\text{pdf:}\quad f(x) = \begin{cases} 1\,, & 0 \le x \le 1 \\ 0\,, & \text{otherwise} \end{cases}$$

The density function is shown below:



The expected value of Ri, is

$$E(R) = \int_0^1 x\,dx = [x^2/2]_0^1 = 1/2$$

The variance is given by 0

$$V(R) = \int_0^1 x^2 dx - [E(R)]^2$$

$$= [x^3/3]_0^1 - (1/2)^2 = 1/3 - 1/4$$

$$= 1/12$$

Some consequences of the uniformity and independence properties are the following:
1. If the interval (0, 1) is divided into n classes, or subintervals of equal length, the expected number of observations m each interval ii N/n where A' is the total number of observations.
2. The probability of observing a value in a particular interval is of the previous values drawn.

5.2 Generation of Pseudo-Random Numbers

Pseudo means false, so false random numbers are being generated. The goal of any generation scheme, is to produce a sequence of numbers between zero and 1 which simulates, or initiates, the ideal properties of uniform distribution and independence as closely as possible. When generating pseudo-random numbers, certain problems or errors can occur. These errors, or departures from ideal randomness, are all related to the properties stated previously. **Some examples include the following**

1) The generated numbers may not be uniformly distributed.

2) The generated numbers may be discrete -valued instead continuous valued

3) The mean of the generated numbers may be too high or too low.

4) The variance of the generated numbers may be too high or low

5) There may be dependence.

The following are examples:

a) Autocorrelation between numbers.

b) Numbers successively higher or lower than adjacent numbers.

c) Several numbers above the mean followed by several numbers below the mean.

Usually, random numbers are generated by a digital computer as part of the simulation. Numerous methods can be used to generate the values. In selecting among these methods, or routines, there are a number of important considerations.

1. The routine should be **fast.** The total cost can be managed by selecting a computationally efficient method of random-number generation.
2. The routine should be **portable** to different computers, and ideally to different programming languages .This is desirable so that the simulation program produces the same results wherever it is executed.
3. The routine should have a sufficiently **long cycle.** The cycle length, or period, represents the length of the random-number sequence before previous numbers begin to repeat themselves in an earlier order. Thus, if 10,000 events are to be generated, the period should be many times that long.
A special case cycling is degenerating. A routine degenerates when the same random numbers appear repeatedly. Such an occurrence is certainly unacceptable. This can happen rapidly with some methods.
4. The random numbers should be **replicable.** Given the starting point (or conditions), it should be possible to generate the same set of random numbers, completely independent of the system that is being simulated. This is helpful for debugging purpose and is a means of facilitating comparisons between systems.
5. Most important, and as indicated previously, the generated random numbers should closely approximate the ideal statistical properties of **uniformity and independences**

5.3 Techniques for Generating Random Numbers

5.3.1 The linear congruential method

It widely used technique, initially proposed by Lehmer [1951], produces a sequence of integers, X1, X2,... between zero and m — 1 according to the following recursive relationship:

$$X_{i+1} = (aX_i + c) \bmod m, \quad i = 0, 1, 2.... \quad (7.1)$$

The initial value **X0** is called the seed, **a** is called the constant multiplier, **c** is the increment, and **m** is the modulus.
If $c \neq 0$ in Equation (7.1), the form is called the **mixed congruential method.** When $c = 0$, the form is known as the **multiplicative congruential method**.

The selection of the values for a, c, m and X0 drastically affects the statistical properties and the cycle length. An example will illustrate how this technique operates.

@I KE@=FHDCO ;@IJH@9KJ<; FE J?< @EJ<><HI % JF D@ Q '$ 3F I<< ?FM J?@I H<IKCJ :8E 9< KI<; JF =FHD :FD9@E<; ><E<H8JFHI" C<J 5@"& " 5@" ' "$$$" 5 @"B 9< J?< J? FKJGKJ =HFD B ;@==<H<EJ DKCJ@GC@:8J@L< :FE>HK<EJ@8C ><E<H8JFHI" M?<H< J?< J? ><E<H8JFH ?8I GH@D< DF;KCKI D@" 8E; J?< DKCJ@GC@<H 8@ @I :?FI<E IF J?8J J?< G<H@F; @I D@ Q &$ 3?<E J?< @ J? ><E<H8JFH @I GHF;K:@E> @EJ<><HI 5@"@ J?8J 8H< 8GGHFN@D8J<CO KE@=FHDCO ;@IJH@9KJ<; FE & JF D@ # &" 8E; 4@"@ , 5@"@ Q & @I 8GGHFN@D8J<CO KE@=FHDCO ;@IJH@9KJ<; FE % JF D@ # '$ 1 . :KO<H 6&*))7 J?<H<=FH< IK>><IJI :FD9@E<; ><E<H8JFHI F= J?< =FHD

5.4 Tests for Random Numbers

1. ***Frequency test***. Uses the Kolmogorov-Smirnov or the chi-square test to compare the distribution of the set of numbers generated to a uniform distribution.
2. ***Autocorrelation test***. Tests the correlation between numbers and compares the sample correlation to the expected correlation of zero.

5.4.1 Frequency Tests

A basic test that should always be performed to validate a new generator is the test of uniformity. Two different methods of testing are available.

1. Kolmogorov-Smirnov(KS test) and

2. Chi-square test.

• Both of these tests measure the degree of agreement between the distribution of a sample of generated random numbers and the theoretical uniform distribution.

• Both tests are on the null hypothesis of no significant difference between the sample distribution and the theoretical distribution.

1. The Kolmogorov-Smirnov test. This test compares the continuous cdf, F(X), of the uniform distribution to the empirical cdf, SN(x), of the sample of N observations. By definition,

F(x) = x, 0 ≤ x ≤ 1

If the sample from the random-number generator is R1 R2, ,..., RN, then the empirical cdf, SN(x), is defined by

$$S_n(x)\frac{\text{number of } R1, R2, \ldots, Rn \text{ which are} \le x}{N}$$

The Kolmogorov-Smirnov test is based on the largest absolute deviation between F(x) and SN(X) over the range of the random variable. That is. it is based on the statistic ***D = max |F(x) -SN(x)|*** *For testing against a uniform cdf, the test procedure follows these steps:*

Step 1: *Rank the data from smallest to largest. Let R (i) denote the i th smallest observation, so that*

$$R(1) \le R(2) \le \ldots \le R(N)$$

Step 2: *Compute*

$$D^{+} = \max_{1 \le i \le n} \left\{ \frac{i}{N} - R_{(i)} \right\}$$

$$D^{-} = \max_{1 \le i \le n} \left\{ R_{(i)} - \frac{i-1}{N} \right\}$$

Step 3: *Compute D = max (D+, D-).*

Step 4: *Determine the critical value,* ***Dα,*** *from* ***Table A.8*** *for the specified significance level α and the given sample size N.*

Step 5:

$D \le D_{\alpha}$ Accept: No Difference between **S_N(x) and F(x)**

$D > D_{\alpha}$ Reject: No Difference between **S_N(x) and F(x)**

We conclude that no difference has been detected between the true distribution of {R1, R2,... RN} and the uniform distribution.

EXAMPLE 6: *Suppose that the five numbers* ***0.44, 0.81, 0.14, 0.05, 0.93*** *were generated, and it is desired to perform a test for uniformity using the Kolmogorov-Smirnov test with a level of significance* ***α of 0.05.***

Step 1: *Rank the data from smallest to largest. 0.05, 0.14, 0.44, 0.81, 0.93*

Step 2: *Compute* ***D+ and D-***

	R_i	$\frac{i}{N}$	$D^{+} = \max_{1 \le i \le n} \left\{ \frac{i}{N} - R_{(i)} \right\}$	$D^{-} = \max_{1 \le i \le n} \left\{ R_{(i)} - \frac{i-1}{N} \right\}$
1	0.05	0.20	0.15	0.05
2	0.14	0.40	0.26	~
3	0.44	0.60	0.16	0.04
4	0.81	0.80	~	0.21
5	0.93	1.00	0.07	0.13

Step3: Compute D = max (D+, D-)

. D=max (0.26, 0.21) = 0.26

Step 4: Determine the critical value, Dα, from Table A.8 for the specified significance level α and the given sample size N. **Here α=0.05, N=5 then value of Dα = 0.565**

Step 5: Since the computed value, 0.26 is less than the tabulated critical value, 0.565,

the hypothesis of no difference between the distribution of the generated numbers and the uniform distribution is not rejected.

compare F(x) with Sn(X)

2. The chi-square test.

The chi-square test uses the sample statistic

$$\chi_0^2 = \sum_{i=0}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Where, Oi is observed number in the i th class

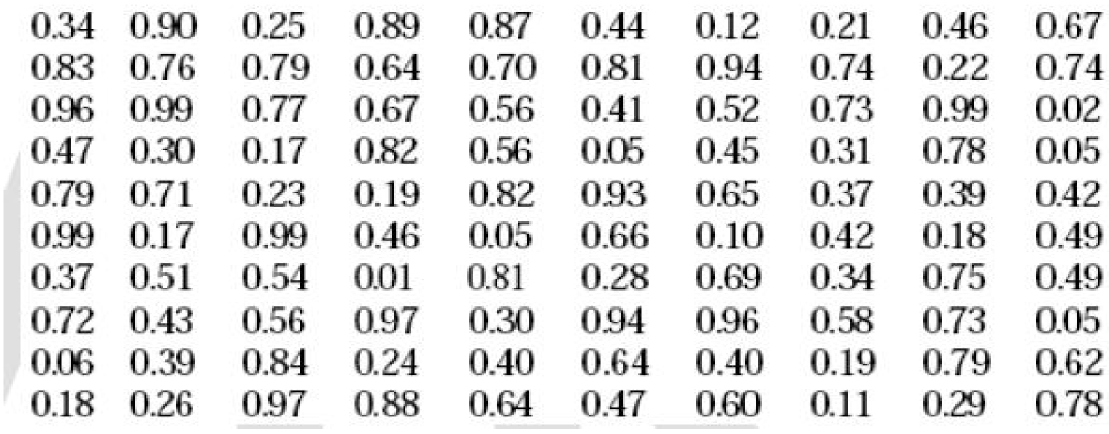
Ei is expected number in the i th class,

N – No. of observation

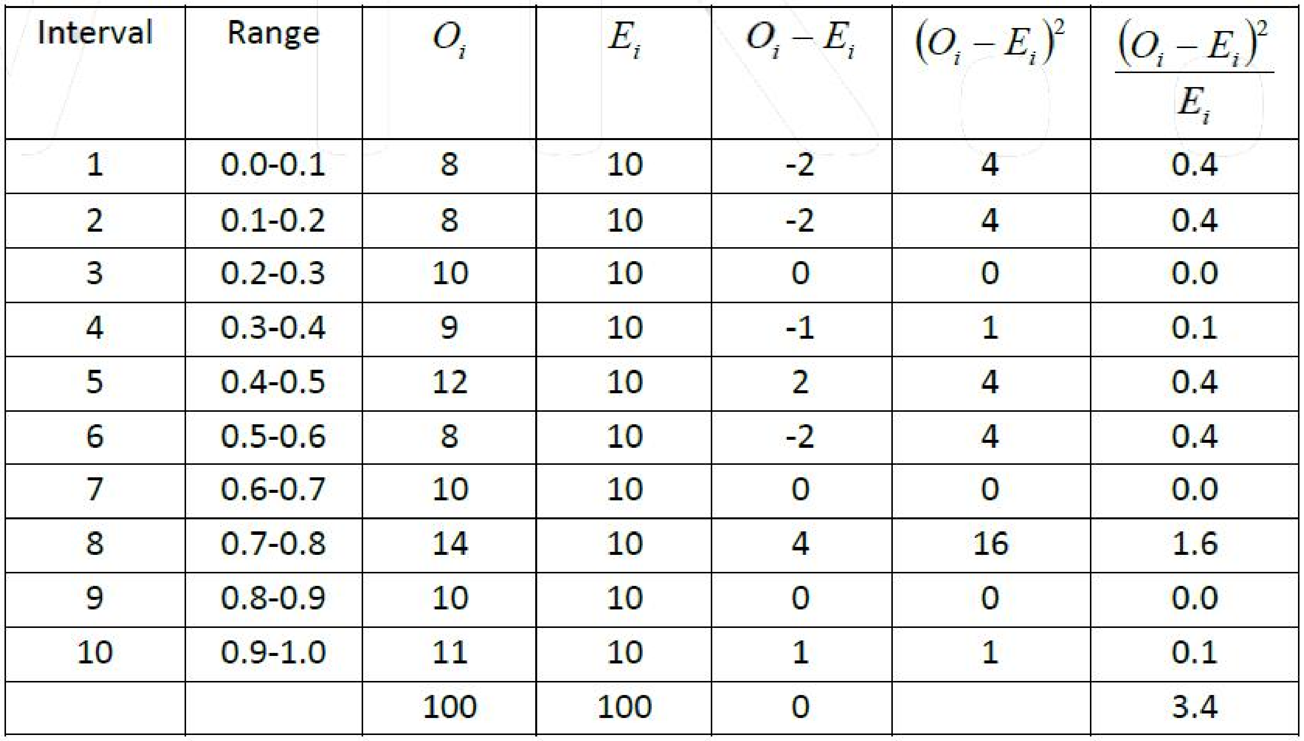
n – No. of classes

Note: sampling distribution of χ_0^2 approximately the chi square has n-1 degrees of freedom

Example 7: *Use the chi-square test with α = 0.05 to test whether the data shown below are uniformly distributed. The test uses n = 10 intervals of equal length, namely [0, 0.1), [0.1, 0.2)... [0.9, 1.0).* ***(REFER TABLE A.6)***



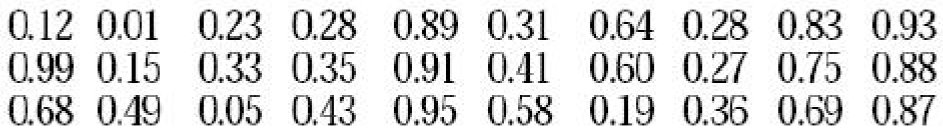
0.34	0.90	0.25	0.89	0.87	0.44	0.12	0.21	0.46	0.67
0.83	0.76	0.79	0.64	0.70	0.81	0.94	0.74	0.22	0.74
0.96	0.99	0.77	0.67	0.56	0.41	0.52	0.73	0.99	0.02
0.47	0.30	0.17	0.82	0.56	0.05	0.45	0.31	0.78	0.05
0.79	0.71	0.23	0.19	0.82	0.93	0.65	0.37	0.39	0.42
0.99	0.17	0.99	0.46	0.05	0.66	0.10	0.42	0.18	0.49
0.37	0.51	0.54	0.01	0.81	0.28	0.69	0.34	0.75	0.49
0.72	0.43	0.56	0.97	0.30	0.94	0.96	0.58	0.73	0.05
0.06	0.39	0.84	0.24	0.40	0.64	0.40	0.19	0.79	0.62
0.18	0.26	0.97	0.88	0.64	0.47	0.60	0.11	0.29	0.78



Interval	Range	O_i	E_i	$O_i - E_i$	$(O_i - E_i)^2$	$\frac{(O_i - E_i)^2}{E_i}$
1	0.0-0.1	8	10	-2	4	0.4
2	0.1-0.2	8	10	-2	4	0.4
3	0.2-0.3	10	10	0	0	0.0
4	0.3-0.4	9	10	-1	1	0.1
5	0.4-0.5	12	10	2	4	0.4
6	0.5-0.6	8	10	-2	4	0.4
7	0.6-0.7	10	10	0	0	0.0
8	0.7-0.8	14	10	4	16	1.6
9	0.8-0.9	10	10	0	0	0.0
10	0.9-1.0	11	10	1	1	0.1
		100	100	0		3.4

5.4.2 Tests for Auto-correlation

The tests for auto-correlation are concerned with the dependence between numbers in a sequence. The list of the 30 numbers appears to have the effect that every 5th number has a very large value. If this is a regular pattern, we can't really say the sequence is random.



0.12	0.01	0.23	0.28	0.89	0.31	0.64	0.28	0.83	0.93
0.99	0.15	0.33	0.35	0.91	0.41	0.60	0.27	0.75	0.88
0.68	0.49	0.05	0.43	0.95	0.58	0.19	0.36	0.69	0.87

The test computes the auto-correlation between every m numbers (m is also known as the lag) starting with the ith number. Thus the autocorrelation ρ_{im} between the following numbers would be of interest.

2.Random Variate Generation TECHNIQUES:

d d d

d r d

d d d d d d d d dmoymd d d d d vq w ssd

d vd d d d d d d d s

d d d d d d d d d s

2.1 Inverse Transform Technique

d d d d d d d d

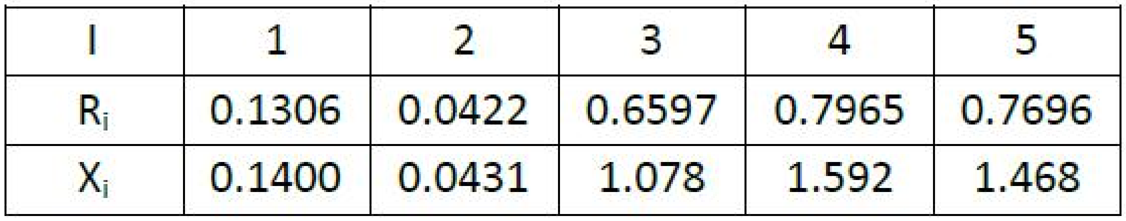
d qd d qd d d d d d s

2.1.1 Exponential Distribution

d d qd d d d dm n

d

d d d dm md d



I	1	2	3	4	5
R_i	0.1306	0.0422	0.6597	0.7965	0.7696
X_i	0.1400	0.0431	1.078	1.592	1.468

Uniform Distribution :

Consider a random variable X that is uniformly distributed on the interval [a, b]. A reasonable guess for generating X is given by

$$X = a + (b - a)R \quad \text{……….5.5}$$

[Recall that R is always a random number on (0,1).

The pdf of X is given by

$$f(x) = \begin{cases} 1/(b-a), & a \le x \le b \\ 0, & \text{otherwise} \end{cases}$$

The derivation of Equation (5..5) follows steps 1 through 3 of Section 5.1.1:

Step 1. The cdf is given by

$$F(x) = \begin{cases} 0, & x < a \\ (x-a)/(b-a), & a \le x \le b \\ 1, & x > b \end{cases}$$

Step 2. Set $F(X) = (X - a)/(b - a) = R$

Step 3. Solving for X in terms of R yields

$$X = a + (b - a)R,$$

which agrees with Equation (5.5).

Weibull Distribution:

The weibull distribution was introduce for test the time to failure of the machine or electronic components. The location of the parameters V is set to 0.

$$f(x) = \begin{cases} \frac{\beta}{\alpha^\beta} x^{\beta-1} e^{-(x/\alpha)^\beta}, & x \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

where α>0 and β>0 are the scale and shape of parameters.

Steps for Weibull distribution are as follows:

step 1: The cdf is given by

step2 :set f(x)=R

$$1 - e^{-(X/\alpha)^\beta} = R.$$

step 3:Solving for X in terms of R yields.

$$X = \alpha[-\ln(1-R)]^{1/\beta}$$

<u>Empirical continuous distribution:</u>

Respampling of data from the sample data in systamtic manner is called empirical continuos distribution.

Step1:Arrange data for smallest to largest order of interval

x(i-1)<x<X(i) i=0,1,2,3….n

Step2:Compute probability 1/n

Step3:Compute cumulative probability i.e i/n   where n is interval

step4:calculate a slope i.e

without frequency   ai=x(i)-x(i-1)/(1/n)

with frequency ai= x(i)-x(i-1)/(c(i)-c(i-1)   where c(i) is cumulative probability

2.1 Acceptance-Rejection technique

- Useful particularly when inverse cdf does not exist in closed form
- Illustration: To generate random variants, *X* ~ U(*1/4*, *1*)
- Procedures:

Step 1: Generate a random number R ~ U [0, 1]

Step 2a: If R ≥ ¼, accept X=R.

Step 2b: If R < ¼, reject R, return to Step 1

- R does not have the desired distribution, but R conditioned (R’) on the event {R ³ ¼} does.
- Efficiency: Depends heavily on the ability to minimize the number of rejections.

2.1.1 Poisson Distribution *A Poisson random variable, N, with mean a > 0 has pmf*

$$p(n) = P(N = n) = \frac{e^{-\alpha}\alpha^{n}}{n!},\ \ n = 0, 1, 2, \ldots$$

- N can be interpreted as number of arrivals from a Poisson arrival process during one unit of time

Then time between the arrivals in the process are exponentially distributed with rate α.

Thus there is a relationship between the (discrete) Poisson distribution and the (continuous) exponential distribution, namely

Unit-5

Random Number Generation & Random Variate Generation

Problems :

① Generate a sequence of 5 integer Random number with $a=19$, $m=100$, $x_0=63$, $\boxed{c=1}$ (mixed Linear Congruential Method)

Solution : $\boxed{X_{i+1} = (a x_i + c) \bmod m,}$ $i=0 \ldots m-1$

→ $i=0$ $X_1 = (19 \times 63+1) \bmod 100$
$= 1198 \bmod 100 = 98$

$i=1$ $\boxed{R_i = \frac{x_i}{m}, \quad i=1 \ldots m}$

$R_1 = \frac{98}{100} = 0.98$

→ $i=1$ $X_2 = (19 \times 98+1) \bmod 100 = 63$

$i=2$ $R_2 = \frac{63}{100} = 0.63$

→ $i=2$ $X_3 = (19 \times 63+1) \bmod 100 = 98$

$i=3$ $R_3 = 98/100 = 0.98$

→ $i=3$ $X_4 = (19 \times 98+1) \bmod 100 = 63$

$i=4$ $R_4 = 63/100 = 0.63$

→ $i=4$ $X_5 = (19 \times 63+1) \bmod 100 = 98$

$i=5$ $R_5 = 98/100 = 0.98$

∴ Random numbers are 0.98, 0.63, 0.98, 0.63, 0.98

(2) Using Multiplicative Congruential Method, Generate sequence of 5 integer number where $a = 24$, $m = 100$, seed value $= 64$.

Solution:

Given $a = 24$

$m = 100$

$x_0 = 64$

$c = 0$

$X_{i+1} = (aX_i + c) \bmod m, \quad i = 0 \dots m-1$

$R_i = \frac{X_i}{m}, \quad i = 1 \dots m$

$X_1 = (24 \times 64 + 0) \bmod 100 = 36$

$R_1 = 36/100 = \underline{\underline{0.36}}$

$X_2 = (24 \times 36 + 0) \bmod 100 = 64$

$R_2 = 64/100 = \underline{\underline{0.64}}$

$X_3 = (24 \times 64 + 0) \bmod 100 = 36$

$R_3 = 36/100 = \underline{\underline{0.36}}$

$X_4 = (24 \times 36 + 0) \bmod 100 = 64$

$R_4 = 64/100 = \underline{\underline{0.64}}$

$X_5 = (24 \times 64 + 0) \bmod 100 = 36$

$R_5 = 36/100 = \underline{\underline{0.36}}$

$\therefore$ Random numbers are 0.36, 0.64, 0.36, 0.64, 0.36

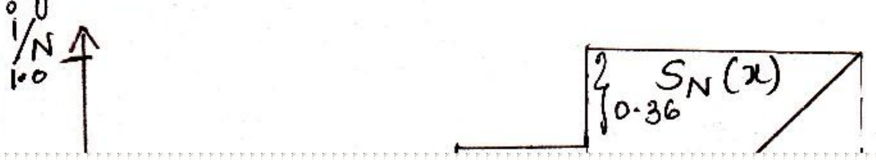
$\therefore$ $D = \max(D^+, D^-) \Rightarrow \max(0.37, 0.18) = 0.37$

from table A8, $D_{\alpha}, N = D_{0.05}, 6 = 0.521$

$\therefore$ $D \leq D_{\alpha} = 0.37 \leq 0.521$

$\therefore$ Accepted Null hypothesis
hence random numbers are uniformly distributed.

Graph :



Solution:

given, $\alpha = 0.05$, $n = 10$, $N = 100$

$$E_i = \frac{N}{n} = \frac{100}{10} = 10$$

interval	O_i	$O_i - E_i$	$(O_i - E_i)^2$	E_i	$X_0^2 = \sum \frac{(O_i - E_i)^2}{E_i}$
1	8	-2	4	10	0.4
2	8	-2	4	10	0.4
3	10	0	0	10	0
4	9	-1	1	10	0.1
5	12	2	4	10	0.4
6	8	-2	4	10	0.4
7	10	0	0	10	0
8	14	4	16	10	1.6
9	10	0	0	10	0
10	11	1	1	10	0.1
Sum	100				3.4

$\therefore X_0^2 = 3.4$

from table A6, $X^2_{\alpha, n-1} = X^2_{0.05, 9} = 16.9$

$\therefore X_0^2 \leq X^2_{\alpha, n-1} = 3.4 \leq 16.9$

$\therefore$ Accepted null hypothesis

(6) use <u>chi-square Test</u> with $\alpha = 0.05$ where $n = 10$, intervals of equal length. sample data are given below:

0.34, 0.90, 0.25, 0.89, 0.87, 0.44, 0.12, 0.21, 0.46, 0.67,
0.83, 0.76, 0.79, 0.64, 0.70, 0.81, 0.94, 0.74, 0.22, 0.74,
0.96, 0.99, 0.77, 0.67, 0.56, 0.41, 0.52, 0.73, 0.99, 0.02
0.47, 0.30, 0.17, 0.82, 0.56, 0.05, 0.45, 0.31, 0.78, 0.05
0.79, 0.71, 0.23, 0.19, 0.82, 0.93, 0.65, 0.37, 0.39, 0.4
0.10, 0.17, 0.10, 0.46, 0.05, 0.66, 0.10, 0.42, 0.18, 0.49,
0.37, 0.51, 0.54, 0.01, 0.81, 0.28, 0.69, 0.34, 0.75, 0.49,
0.72, 0.43, 0.56, 0.97, 0.30, 0.94, 0.96, 0.58, 0.73, 0.05,
0.06, 0.39, 0.84, 0.24, 0.40, 0.64, 0.40, 0.19, 0.79, 0.62,
0.18, 0.26, 0.97, 0.88, 0.64, 0.47, 0.60, 0.11, 0.29, 0.78

<u>Solution</u>: given, $\alpha = 0.05$, $n = 10$, $N = 100$

$E_i = N/n = 100/10 = 10$

interval	O_i	E_i	$O_i - E_i$	$(O_i - E_i)^2$	$X_0^2 = \sum \frac{(O_i - E_i)^2}{E_i}$
0.01-0.10	10	10	0	0	0
0.11-20	8	10	-2	4	0.4
0.21-30	10	10	0	0	0
0.31-40	9	10	-1	1	0.1
0.41-50	12	10	2	4	0.4
0.51-60	8	10	-2	4	0.4
0.61-70	10	10	0	0	0
0.71-80	14	10	4	16	1.6
0.81-90	10	10	0	0	0
0.91-1.00	9	10	-1	1	0.1

$X_3 = \frac{-1}{1} \ln(1-0.10) = \underline{\underline{0.1053}}$

$X_4 = \frac{-1}{1} \ln(1-0.50) = \underline{\underline{0.6931}}$

$X_5 = \frac{-1}{1} \ln(1-0.60) = \underline{\underline{0.9162}}$

9) Generate 5 Random Variation using Uniform Distribution technique with interval $0.3 \le x \le 2$.
consider sample data, 0.30, 0.25, 0.80, 0.75, 2.5

Solution: given, $a = 0.3$ $b = 2$

$$\boxed{X_i = a + (b-a) R_1 \quad , i = 1 \dots n}$$

i	1	2	3	4	5
R_i	0.30	0.25	0.80	0.75	2.5
X_i	0.81	0	1.66	1.575	1

$X_1 = 0.3 + (2-0.3)\,0.3 = \underline{\underline{0.81}}$

$X_2 = 0.25 < 0.3 = \underline{\underline{0}}$

$X_3 = 0.3 \le 0.8 \le 2 \Rightarrow 0.3 + (2-0.3)\,0.80 = \underline{\underline{1.66}}$

$X_4 = 0.3 \le 0.75 \le 2 \Rightarrow 0.3 + (2-0.3)\,0.75 = \underline{\underline{1.575}}$

$X_5 = 2.5 > 2 \Rightarrow \underline{\underline{1}}$

(13) 10m * Generate 3 poisson variate with mean 0.2. consider random numbers 0.4357, 0.4146, 0.8353, 0.9952, 0.8004.
vvimp

Solution : Steps : (2marks)

Step 0: Set $n=0$, $p=1$

2: $R_1 = 0.4357$ $\quad P = P \cdot R_1 = 1 \times 0.4357 = 0.4357$

$e^{-\alpha} = e^{-0.2} = 0.8187$

3: $P < e^{-\alpha} = 0.4357 < 0.8187$ $\boxed{\therefore \text{Accept } N=0}$

Step 1: $n=0$, $p=1$

2: $R_2 = 0.4146$ $\quad P = P \cdot R_2 = 1 \times 0.4146 = 0.4146$

3: $0.4146 < 0.8187$ $\boxed{\therefore \text{Accept } N=0}$

Step 1: $n=0$, $p=1$

2: $R_3 = 0.8353$ $\quad p = 1 \times 0.8353 = 0.8353$

3: $0.8353 < 0.8187$ $\boxed{\therefore \text{Reject } n=1}$ $\quad p = 0.8353.$

$n=1$, $p = 0.8353$

Step 2: $R_4 = 0.9952$ $\quad p = P \cdot R_4 = 0.8353 \times 0.9952 = 0.8312$

3: $0.8312 < 0.8187$ $\boxed{\therefore \text{Reject } n=2}$

$n=2$, $p = 0.8312$

Step 2: $R_5 = 0.8004$ $\quad p = 0.8312 \times 0.8004 = 0.6652$

3: $0.6652 < 0.8187$ $\boxed{\therefore \text{Accept } N=2}$

*table : (8 marks)

n	P	R_{n+1}	$P = P \cdot R_{n+1}$	$P < e^{-\alpha}$ A/R	$N = n$.
0	1	0.4357	0.4357	$0.4357 < 0.8187$ Accept	$N=0$
0	1	0.4146	0.4146	$0.4146 < 0.8187$ Acept	$N=0$
0	1	0.8353	0.8353	$0.8353 < 0.8187$ Reject	–
1	0.8353	0.9952	0.8312	$0.8312 < 0.8187$ Reject	–
2	0.8312	0.8004	0.6652	$0.6652 < 0.8187$ Acept	$N=2$

∴ the 3 poisson variates are $N=0, N=0, N=2$.

(14) Generate 5 poisson variate with mean $= 0.2$

Solution : Steps :

$e^{-\alpha} = e^{-0.2} = 0.8187$.

$n = 5$

*) If random numbers are not given, then the random numbers are selected so that it satisfies following conditions : If x is random number then, $\frac{1}{4} \leq x \leq 1$ & $x \leq e^{-\alpha}$ then accepted, else reject.

Step 1: $n=0$, $p=1$

2: $R_1 = 0.3568$ $\quad p = p \times R_1 = 1 \times 0.3568 = 0.3568$

3: $0.3568 \leq 0.8187$ & $0.25 \leq 0.3568 \leq 1$ $\boxed{\therefore \text{Accept } N=0}$

Step 1) $n=0$ $\quad p=1$

2: $R_2 = 0.4123$ $\quad p = 1 \times 0.4123 = 0.4123$

unit 6: INPUT MODELING

6. INPUT MODELING

- Input data provide the driving force for a simulation model. In the simulation of a queuing system, typical input data are the distributions of time between arrivals and service times.
- For the simulation of a reliability system, the distribution of time-to=failure of a component is an example of input data.

There are four steps in the development of a useful model of input data:

- Collect data from the real system of interest. This often requires a substantial time and resource commitment. Unfortunately, in some situations it is not possible to collect data
- Identify a probability distribution to represent the input process. When data are available, this step typically begins by developing a frequency distribution, or histogram, of the data.
- Choose parameters that determine a specific instance of the distribution family. When data are available, these parameters may be estimated from the data.
- Evaluate the chosen distribution and the associated parameters for good-of- fit. Goodness-of-fit may be evaluated informally via graphical methods, or formally via statistical tests. The chisquare and the Kolmo-gorov-Smirnov tests are standard goodness-of-fit tests. If not satisfied that the chosen distribution is a good approximation of the data, then the analyst returns to the second step, chooses a different family of distributions, and repeats the procedure. If several iterations of this procedure fail to yield a fit between an assumed distributional form and the collected data

<u>6.1 Data Collection</u>

- Data collection is one of the biggest tasks in solving real problem. It is one of the most important and difficult problems in simulation. And even if when data are available, they have rarely been recorded in a form that is directly useful for simulation input modeling.

The following suggestions may enhance and facilitate data collection, although they are not all – inclusive.

1. A useful expenditure of time is in planning. This could begin by a practice or pre observing session. Try to collect data while pre-observing.
2. Try to analyze the data as they are being collected. Determine if any data being collected are useless to the simulation. There is no need to collect superfluous data.
3. Try to combine homogeneous data sets. Check data for homogeneity in successive time periods and during the same time period on successive days.
4. Be aware of the possibility of data censoring, in which a quantity of interest is not observed in its entirety. This problem most often occurs when the analyst is interested in the time required to complete some process (for example, produce a part, treat a patient, or have a component fail), but the process begins prior to, or finishes after the completion of, the observation period.
5. To determine whether there is a relationship between two variables, build a scatter diagram.
6. Consider the possibility that a sequence of observations which appear to be independent may possess autocorrelation. Autocorrelation may exist in successive time periods or for successive customers.
7. Keep in mind the difference between input data and output or performance data, and be sure to collect input data. Input data typically represent the uncertain quantities that are largely beyond the control of the system and will not be altered by changes made to improve the system.

6.2 Identifying the Distribution with Data.

- In this section we discuss methods for selecting families of input distributions when data are available.

6.2.1 Histogram

- A frequency distribution or histogram is useful in identifying the shape of a distribution. A histogram is constructed as follows:

1. Divide the range of the data into intervals (intervals are usually of equal width;

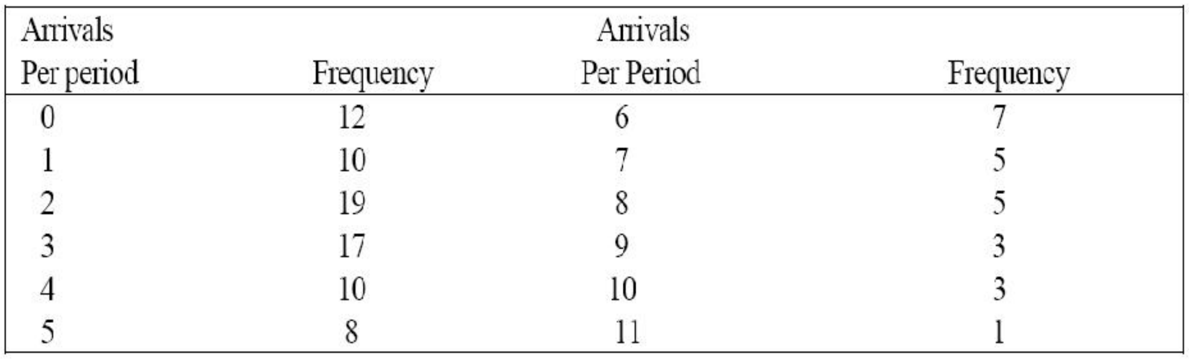
however, unequal widths however, unequal width may be used if the heights of the frequencies are adjusted).

2. Label the horizontal axis to conform to the intervals selected.
3. Determine the frequency of occurrences within each interval.
4. Label the vertical axis so that the total occurrences can be plotted for each interval.
5. Plot the frequencies on the vertical axis.

- If the intervals are too wide, the histogram will be coarse, or blocky, and its shape and other details will not show well. If the intervals are too narrow, the histogram will be ragged and will not smooth the data.
- The histogram for continuous data corresponds to the probability density function of a theoretical distribution.

Example 6.2 : The number of vehicles arriving at the northwest corner of an intersection in a 5 min period between 7 A.M. and 7:05 A.M. was monitored for five workdays over a 20-week period. Table shows the resulting data. The first entry in the table indicates that there were 12:5 min periods during which zero vehicles arrived, 10 periods during which one vehicles arrived, and so on,

Table 6:1 Number of Arrivals in a 5 Minute period



Arrivals Per period	Frequency	Arrivals Per Period	Frequency
0	12	6	7
1	10	7	5
2	19	8	5
3	17	9	3
4	10	10	3
5	8	11	1

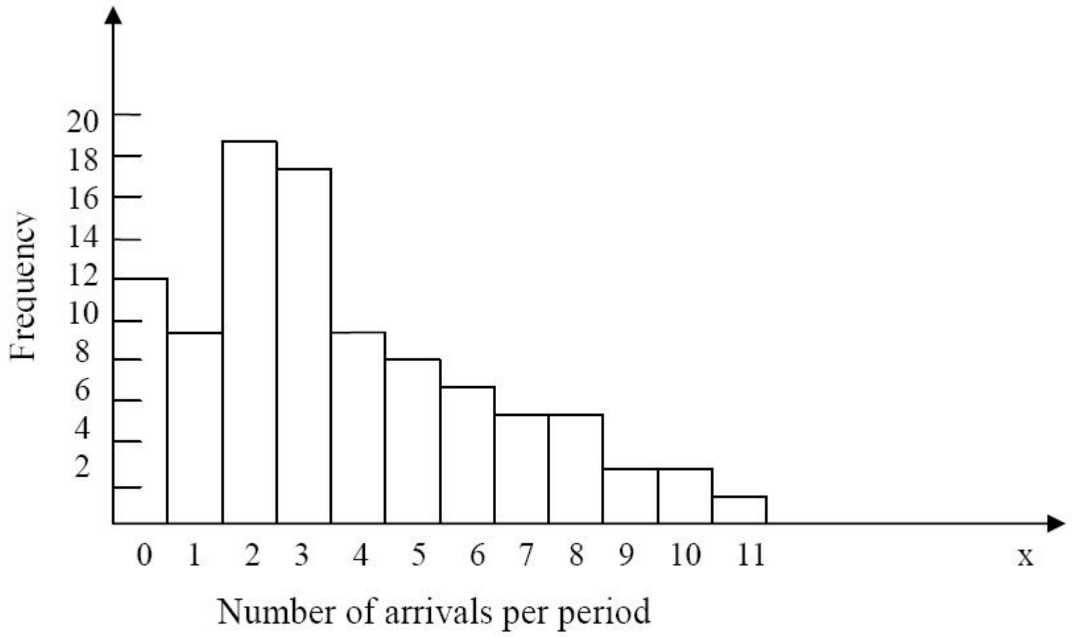


Fig 6.2 Histogram of number of arrivals per period.

<u>6.2.2 Selecting the Family of Distributions</u>

- Additionally, the shapes of these distributions were displayed. The purpose of preparing histogram is to infer a known pdf or pmf. A family of distributions is selected on the basis of what might arise in the context being investigated along with the shape of the histogram.

- Thus, if interarrival-time data have been collected, and the histogram has a shape similar to the pdf in Figure 5.9.the assumption of an exponential distribution would be warranted.

- Similarly, if measurements of weights of pallets of freight are being made, and the histogram appears symmetric about the mean with a shape like that shown in Fig 5.12, the assumption of a normal distribution would be warranted.

- The exponential, normal, and Poisson distributions are frequently encountered and are not difficult to analyze from a computational standpoint. Although more difficult to analyze, the gamma and Weibull distributions provide array of shapes, and should not be overlooked when modeling an underlying probabilistic process. Perhaps an exponential distribution was assumed, but it was found not to fit the data. The next step would be to examine where the lack of fit occurred.

- If the lack of fit was in one of the tails of the distribution, perhaps a gamma or Weibull distribution would more adequately fit the data.

- Literally hundreds of probability distributions have been created, many with some specific physical process in mind. One aid to selecting distributions is to use the physical basis of the distributions as a guide. Here are some examples:

<u>6.2.3 Quantile-Quantile Plots</u>

- Further, our perception of the fit depends on widths of the histogram intervals. But even if the intervals are well chosen, grouping of data into cells makes it difficult to compare a histogram to a continues probability density function

- If X is a random variable with cdf F, then the q-quintile of X is that y such that F(y) = P(X < y) = q, for 0 < q < 1. When F has an invererse, we write y = F-1(q).

- Now let {Xi, i = 1, 2,...,n} be a sample of data from X. Order the observations from the smallest to the largest, and denote these as {yj, j =1,2 ,,,n}, where y1 < y2 < ….. < y_n- Let j denote the ranking or order number. Therefore, j = 1 for the smallest and j = n for the largest. The q-q plot is based on the fact that y1 is an estimate of the (j — 1/2)/n quantile of X other words,

$$\text{Yj is approximately } F^{-1} \left[\frac{J - \frac{1}{2}}{n} \right]$$

- Now suppose that we have chosen a distribution with cdf F as a possible representation of the distribution of X. If F is a member of an appropriate family of distributions, then a plot of y_j versus F^{-1}((j —1/2)/n) will be approximately a straight line.

6.3 Parameter Estimation

- After a family of distributions has been selected, the next step is to estimate the parameters of the distribution. Estimators for many useful distributions are described in this section. In addition, many software packages—some of them integrated into simulation languages—are now available to compute these estimates.

6.3.1 Preliminary Statistics: Sample Mean and Sample Variance

- In a number of instances the sample mean, or the sample mean and sample variance, are used to estimate of the parameters of hypothesized distribution;

- If the observations in a sample of size n are X1, X2,..., Xn, the sample mean (X) is defined by

$$\overline{X} = \frac{\sum^{n}_{i=1} Xi}{n} \qquad 9.1$$

and the sample variance, s^2 is defined by

$$S^2 = \frac{\sum^{n}_{i=1} Xi^2 - n\overline{X}^2}{n - 1} \qquad 9.2$$

If the data are discrete and grouped in frequency distribution, Equation (9.1) and (.2) can be modified to provide for much greater computational efficiency, The sample mean can be computed by

And the sample variance by

where k is the number of distinct values of X and fj is the observed frequency of the value Xj, of X.

6.3.2 Suggested Estimators

- Numerical estimates of the distribution parameters are needed to reduce the family of distributions to a specific distribution and to test the resulting hypothesis.
- These estimators are the maximum-likelihood estimators based on the raw data. (If the data are in class intervals, these estimators must be modified.)
- The triangular distribution is usually employed when no data are available, with the parameters obtained from educated guesses for the minimum, most likely, and maximum possible value's; the uniform distribution may also be used in this way if only minimum and maximum values are available.

H0: the random variable, X, conforms to the distributional assumption with the parameter(s) given by the parameter estimate(s)

H1 : the random variable X does not conform

- If the distribution being tested is discrete, each value of the random variable should be a class interval, unless it is necessary to combine adjacent class intervals to meet the minimum expected cell-frequency requirement. For the discrete case, if combining adjacent cells is not required,

$Pi = P(X_I) = P(X X_i)$

Otherwise, pi, is determined by summing the probabilities of appropriate adjacent cells.

- If the distribution being tested is continuous, the class intervals are given by $[a_{i-1}, ai)$, , where ai-1 and ai, are the endpoints of the ith class interval. For the continuous case with assumed pdf f(x), or assumed cdf F(x), pi, can be computed By

$Pi= {}_{ai-1}{}^{ai} f(x)\, dx= F(a_i) – F(a_{i\ -1})$

<u>6.4.2 Chi-Square Test with Equal Probabilities</u>

- If a continuous distributional assumption is being tested, class intervals that are equal in probability rather than equal in width of interval should be used.
- Unfortunately, there is as yet no method for deter mining the; probability associated with each interval that maximize the; power of a test o f a given size.

$Ei = n\ p\ i \quad 5$

- Substituting for p i yields $n/k \quad 5$

- and solving for k yields $k \quad n/5$

- The arrival times T1, T1+T2, T1+T2+T3,…..,T1+…..+T50 are obtained by adding interarrival times.
- On a (0,1) interval, the points will be [T1/T, (T1+T2)/T,…..,(T1+….+T50)/T].

6.5 Selecting Input Models without Data

Unfortunately. it is often necessary in practice to develop a simulation model for demonstration purposes or a preliminary study—before any i data are available.) In this case the modeler must be resourceful in choosing input models and must carefully check the sensitivity of results to the models.

Engineering data : Often a product or process has performance ratings pro vided by the manufacturer.

Expert option : Talk to people who are experienced with the procesws or similar processes. Often they can provide optimistic, pessimistic and most likely times.

Physical or conventional limitations : Most real processes have physical limit on performance. Because of company policies, there may be upper limits on how long a process may take. Do not ignore obvious limits or bound: that narrow the range of the input process.

The nature of the process It can be used to justify a particular choice even when no data are available.

6.6 Multivariate and Time-Series Input Models

The random variables presented were considered to be independent of any other variables within the context of the problem. However, variables may be related, and if the variables appear in a simulation model as inputs, the relationship should be determined and taken into consideration.

Step 1. Generate Z_1 and Z_2, independent standard normal random variables.

Step 2. Set $X_1 = \mu_1 + \sigma_1 Z_1$

Step 3. Set $X_2 = \mu_2 + \sigma_2 \left(\rho Z_1 + \sqrt{1-\rho^2} Z_2\right)$

6.7 Time series input model:

If X1,X2..Xn is a sequence of identically distributed,but dependent and convarianc stationary random variables,then there are a number of times series model that can be used to represent the process. The two models that have the characteristics that the autocorrelatrion take the form.

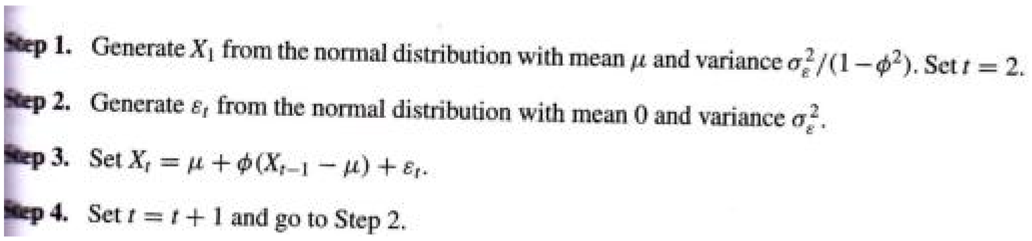
$$\rho_h = \operatorname{corr}(X_t, X_{t+h}) = \rho^h$$

for h=1,2,..n that the log-h autocorrelation decreases geometrically as the lag increases.

AR(1) Model:

consider the time series model

for t=2,3,..n where ε2, ε3 are the independent and identically distributed with men 0 and variance σ^2_ε and -1< ϕ<1. If the initial value x1 is chosen appropriately,then x1,x2..are all normal distributed with mean u and variance



Step 1. Generate X_1 from the normal distribution with mean μ and variance $\sigma_\varepsilon^2/(1-\phi^2)$. Set $t = 2$.

Step 2. Generate ε_t from the normal distribution with mean 0 and variance σ_ε^2.

Step 3. Set $X_t = \mu + \phi(X_{t-1} - \mu) + \varepsilon_t$.

Step 4. Set $t = t + 1$ and go to Step 2.

EAR(1) Model:

Consider the time series model

for t=2,3,..n where ε2, ε3 are the independent and identically distributed with mean and 0< ϕ<1. If the initial value x1 is chosen appropriately, then x1,x2.. are all exponentially distributed with mean and variance

Step 1. Generate X_1 from the exponential distribution with mean $1/\lambda$. Set $t = 2$.

Step 2. Generate U from the uniform distribution on [0, 1]. If $U \leq \phi$, then set

$$X_t = \phi X_{t-1}$$

Otherwise, generate ε_t from the exponential distribution with mean $1/\lambda$ and set

$$X_t = \phi X_{t-1} + \varepsilon_t$$

Step 3. Set $t = t + 1$ and go to Step 2.

Goodness-of-fit Tests

⊙ Chi-Square Test with poisson Assumption

Step 1: Compute poisson distribution using

$$P(x) = \begin{cases} \frac{e^{-\alpha} \alpha^{x}}{x!} \end{cases} \qquad x = 0, 1, 2, \ldots$$

Step 2: Compute expected frequency

$E_i = n . P(i)$ where n is sum of Sample data

Reduce interval i.e

$E_i > 5$

Step 3: Compute Chi-square test i.e

$$X_0^2 = \sum_{i=0}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Step 4: Obtain chi-square test value from table A.6

$$X^2_{\alpha, K-S-1}$$

Step 5: Check hypothesis or Null hypothesis

$X_0^2 \lessgtr X^2_{\alpha, K-S-1}$ Accepted hypothesis / Rejected Null hypothesis

*) Kolmogorov - Smirnov test for exponential distributions

Step 1: Calculate inter arrival points

$$\boxed{R_i = \{ T_1/T, \ (T_1+T_2)/T, \ (T_1+T_2+T_3)/T, \ (T_1+T_2+T_n)/T \}}$$

T is total No's of sample data

T_i - is the sample data

Step 2: Compute

$$\boxed{\begin{array}{l} D^+ = \max\limits_{1 \le i \le n} \{ i/n - R(i) \} \\ D^- = \max\limits_{1 \le i \le n} \{ R(i) - \frac{i-1}{n} \} \end{array}}$$

Step 3: Compute

$$\boxed{D = \max (D^+, D^-)}$$

Step 4: Obtain KS - test value from table A.8

$$\boxed{D_{\alpha, n}}$$

Step 5: Check hypothesis or Null hypothesis

$\boxed{D \le D_{\alpha, n}}$ accepted

UNIT-6 : INPUT MODELLING

I. Chi Square Test using Poisson Assumption

1. Using goodness of fit test, test whether random Nos are uniformly distributed based on poisson assumption with level of significance $\alpha = 0.05$. $\hat{\alpha} = 3.64$. Sample data are :

Interval :	0	1	2	3	4	5	6	7	8	9	10	11
observed Frequency :	12	10	19	17	10	8	7	5	5	3	3	1

⇒ Given : $\alpha = 0.05$

$\hat{\alpha} = 3.64$

$n = 12 + 10 + 19 + 17 + 10 + 8 + 7 + 5 + 5 + 3 + 3 + 1 = 100$

Step 1 : Compute Poisson Distribution

$$P(x) = \frac{e^{-\alpha}\alpha^{x}}{x!} \quad \text{where } x = 0, 1 \ldots, 11$$

$$P(0) = \frac{e^{-3.64} * (3.64)^{0}}{0!} = 0.026$$

$P(1) = 0.096$

$P(2) = 0.174$

$P(3) = 0.211$

$P(4) = 0.192$

$P(5) = 0.140$

$P(6) = 0.085$

$P(7) = 0.044$

$P(8) = 0.020$

$P(9) = 0.008$

$P(10) = 0.003$

$P(11) = 0.001$

Step 2 : Apply Chi Square test with poisson assumption

X_i	O_i	$E_i = n \cdot P_i$	$O_i - E_i$	$(O_i - E_i)^2$	$X_0^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$
0	5 } 15	1.4 } 6.86	8.14	66.26	9.66
1	10	5.46			
2	5	10.71	−5.71	32.60	3.04
3	8	13.93	−5.93	35.16	2.52
4	12	13.65	−1.65	2.72	0.19
5	10	10.71	−0.71	0.50	0.05
6	8 } 20	6.93 } 10.85	9.15	83.72	7.72
7	12	3.92			

Here $k = 6$, $S = 1$ $\quad\quad X_0^2 = 23.18$

Step 3 : Compute level of Significance from Table A6

$X_{0\,\alpha,\,k-S-1}^2 = X_{0\,0.05,\,6-1-1}^2 = 9.49$

Step 4 : Check whether Random No.s are uniformly distributed.

Compare X_0^2 & $X_{0\,0.05,\,4}^2$

$\because 23.18 > 9.49 \Rightarrow$ Random No.s are not uniformly distributed

(6) Step 1 :

$R_{(i)} = \{0.0044, 0.0097, 0.0301, 0.0575, 0.0775, 0.0805, 0.1059, 0.1111, 0.1313, 0.1502\}$

Step 2 :

i	$R_{(i)}$	i/n	$i-1/n$	$D^+ = \max\{\frac{i}{n} - R_{(i)}\}$	$D^- = \max\{R_{(i)} - \frac{i-1}{n}\}$
1	0.0044	0.1	0	0.0956	0.0044
2	0.0097	0.2	0.1	0.1903	~
3	0.0301	0.3	0.2	0.2699	~
4	0.0575	0.4	0.3	0.3425	~
5	0.0775	0.5	0.4	0.4225	~
6	0.0805	0.6	0.5	0.5195	~
7	0.1059	0.7	0.6	0.5941	~
8	0.1111	0.8	0.7	0.6889	~
9	0.1313	0.9	0.8	0.7687	~
10	0.1502	1.0	0.9	0.8498	~

Step 3 :

$D = \max\{D^+, D^-\} = \max\{0.8498, 0.0044\}$

$D = \underline{0.8498}$

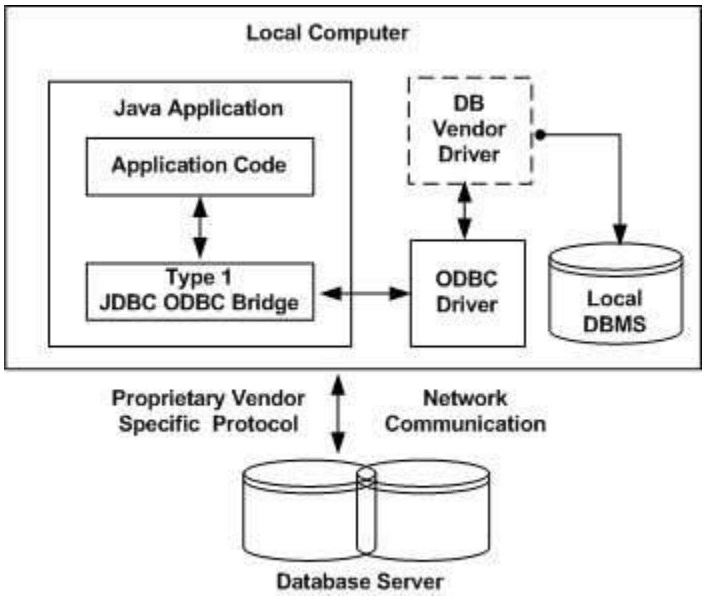
Step 4 :

D_{α}, n from A8 table.

$D_{0.05}, 10 = \underline{0.410}$

Step 5 :

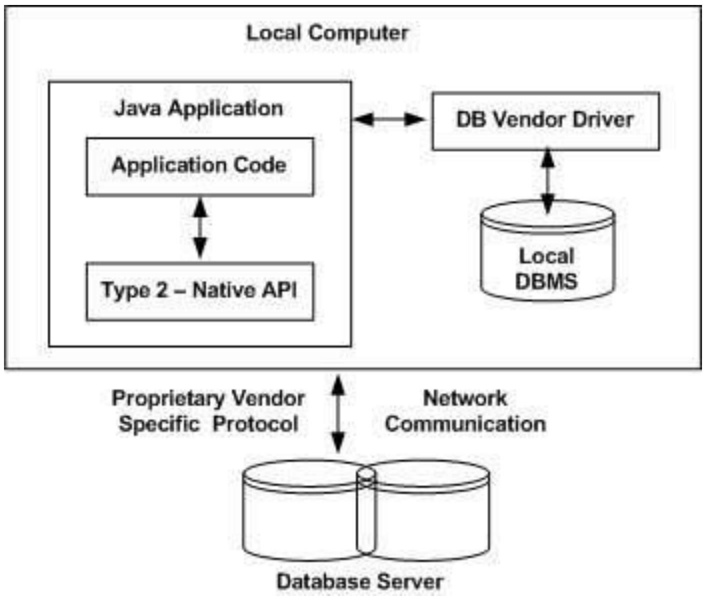
$\therefore$ $0.8498 > 0.410 \Rightarrow$ Random No's are rejected



- A JDBC/ODBC bridge provides JDBC API access through one or more ODBC drivers. Some ODBC native code and in many cases native database client code must be loaded on each client machine that uses this type of driver.
- **The advantage** for using this type of driver is that it allows access to almost any database since the database ODBC drivers are readily available.
- Disadvantages for using this type of driver include the following:
    - Performance is degraded since the JDBC call goes through the bridge to the ODBC driver then to the native database connectivity interface. The results are then sent back through the reverse process
    - Limited Java feature set
    - May not be suitable for a large-scale application

<u>Type Two Driver:Native API/JAVA protocal</u>

- In a Type 2 driver, JDBC API calls are converted into native C/C++ API calls, which are unique to the database. These drivers are typically provided by the database vendors and used in the same manner as the JDBC-ODBC Bridge. The vendor-specific driver must be installed on each client machine.
- If we change the Database, we have to change the native API, as it is specific to a database and they are mostly obsolete now, but you may realize some speed increase with a Type 2 driver, because it eliminates ODBC's overhead.



The Oracle Call Interface (OCI) driver is an example of a Type 2 driver.

Advantages for using this type of driver include the following:

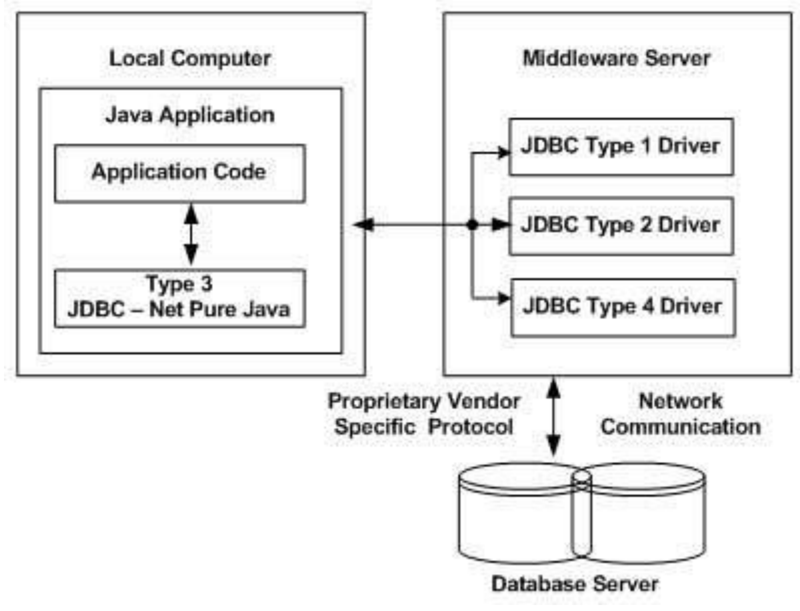
- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge
- Limited Java feature set

Disadvantages for using this type of driver include the following:

- Applicable Client library must be installed
- Type 2 driver shows lower performance than type 3 or 4

<u>Type 3: JDBC-Net pure Java(JDBC PROTOCAL)</u>

- In a Type 3 driver, a three-tier approach is used to access databases. The JDBC clients use standard network sockets to communicate with a middleware application server. The socket information is then translated by the middleware application server into the call format required by the DBMS, and forwarded to the database server.
- This kind of driver is extremely flexible, since it requires no code installed on the client and a single driver can actually provide access to multiple databases.



Advantages for using this type of driver include the following:

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge and Type 2 Drivers
- Advanced Java feature set
- Scalable
- Caching
- Advanced system administration
- Does not require applicable database client libraries

The disadvantage for using this type of driver is that it requires a separate JDBC middleware server to translate specific native-connectivity interface.

Type 4: Pure Java protocal

- In a Type 4 driver, a pure Java-based driver communicates directly with the vendor's database through socket connection. This is the highest performance driver available for the database and is usually provided by the vendor itself.
- This kind of driver is extremely flexible, you don't need to install special software on the client or server. Further, these drivers can be downloaded dynamically.

Advantages for using this type of driver include the following:

- Allows access to almost any database since the databases ODBC drivers are readily available
- Offers significantly better performance than the JDBC/ODBC Bridge and Type 2 Drivers
- Scalable
- Caching
- Advanced system administration
- Superior performance
- Advance Java feature set
- Does not require applicable database client libraries

The disadvantage for using this type of driver is that each database will require a driver

3.A brief overview of the JDBC process:

This process is divided into five steps:

- Loading the jdbc drivers
- Connecting to dbms
- Creating and executing statements
- Processing data returned by dbms
- Terminating the connection with the dbms

```
        S.o.p(e);     }
```

Create and execute sql statements:

- After jdbc driver is loaded and connection is successfully made with the databse.
- Now database is managed by the dbms is to send a sql query to the dbms for processing.
- The createStatement() method is used to create the statement object. The createStatement() method is belongs to connection interface.
- The return value of createStatement() method is the Statement interface.
- The statement object is used to execute queryand return a resultset interface objectthat conatains the response from the dbms.
- The different methods are used to execute the query are as follows:

```
                executeQuery(String)
                executeUpadate(String)
                execute(string)
```

Code snippet :

```
        try
        {
        Class.forName("com.mysql.jdbc.Driver");
        Connection
c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
        Statement s=c.createStatement();
        ResultSet r=s.executeQuery("Select *from emp");
        }
        catch(Exception e)
        {
        S.o.p(e);
        }
```

Process data returned by the dbms:

- ResultSet object is assigned to receive the data from the DBMS after the query processed.
- ResultSet object conatins the method used to intract with the data that is returned by DBMS to the j2ee components.

program to retrieve the data from the database:

```
        import java.sql.*;
        class A
        {
        A()
        {
        try
        {
        Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
        Statement s=c.createStatement();
        ResultSet r=s.executeQuery("Select *from emp");
        while(r.next())
        {
        String name=r.getString(1);
        String usn=r.getString(2);
        System.out.println("name="+name);
        System.out.println("USN="+usn);
        }
        c.close();
        }
        catch(Exception e)
        {
        S.o.p(e);
        }
        }
        public stataic void main(String ar[])
        {
```

```
}
catch(Exception e)
{
Sysetm.out.println(e);
}
```

NOTE:just for reference-Following table lists down the popular JDBC driver names and database URL.

RDBMS	**JDBC driver name**	**URL format**
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql://**hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

```
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

5.The Statement Objects

- Once connection to the databse is opened,the j2ee component creates and sends a query to access data contained in database.
- There are three ways statement object are used:
    - Statement object
    - preparedStatemnt object
    - callableStatement object

Creating Statement Object

- Before you can use a Statement object to execute a SQL statement, you need to create one using the Connection object's createStatement( ) method, as in the following example

    Statement s=c.createStatement();

```
}
catch(Exception e)
{
S.o.p(e);
}
}
Public stataic void main(String ar[])
{
A a1=new A();
}
}
```

<u>The PreparedStatement Objects</u>

- The *PreparedStatement* interface extends the Statement interface, which gives you added functionality with a couple of advantages over a generic Statement object.
- This statement gives you the flexibility of supplying arguments dynamically.

  PreparedStatement p=new PrepareStatement("select name from emp where usn=?");

- The setXXX() methods bind values to the parameters, where XXX represents the Java data type of the value you wish to bind to the input parameter.
  - **setXXX(int,string);**
- First parameter represent the column index and second parameter represent the values that replace the ? mark in the query.
- Next different execut methods of the preparedStatement object are called.

```
import java.sql.*;
class A
{
A()
{
try
{
        Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
  PreparedStatement p=c.PreapareStatement("select name from emp where usn=?");
  p.setSting(2, "12cs001");
  ResultSet r=p.executeQuery();
        while(r.next())
        {
        String name=r.getString(1);
        String usn=r.getString(2);
        System.out.println("name="+name);
        System.out.println("USN="+usn);
        }
        c.close();
        }
        catch(Exception e)
        {
        S.o.p(e);
        }
        }
        public static void main(String ar[])
        {
        A a1=new A();
        }
        }
```

The CallableStatement Objects

- Just as a Connection object creates the Statement and PreparedStatement objects, it also creates the CallableStatement object, which would be used to execute a call to a database stored procedure.
- Three types of parameters exist: IN, OUT, and INOUT. The PreparedStatement object only uses the IN parameter. The CallableStatement object can use all the three.
- Here are the definitions of each −

```
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
        CallableStatement p=c. prepareCall ("Call lastOrderNumber(?)");
        p.registerOutParameter(1,TYPES.VARCHAR);
        p.executeQuery();
        String name=p.getString(1);
        System.out.println("name="+name);
        }
        c.close();
        }
        catch(Exception e)
        {
        S.o.p(e);
        }
        }
        public stataic void main(String ar[])
        {
        A a1=new A();
        }
        }
```

6. ResultSet

A `ResultSet` consists of records. Each records contains a set of columns.

A `ResultSet` can be of a certain type. The type determines some characteristics and abilities of the `ResultSet`.

Scrollable ResultSet:

At the time of writing there are three `ResultSet` types:

1. ResultSet.TYPE_FORWARD_ONLY
2. ResultSet.TYPE_SCROLL_INSENSITIVE
3. ResultSet.TYPE_SCROLL_SENSITIVE

The default type is `TYPE_FORWARD_ONLY`

- `TYPE_FORWARD_ONLY` means that the `ResultSet` can only be navigated forward. That is, you can only move from row 1, to row 2, to row 3 etc. You cannot move backwards in the `ResultSet`.
- `TYPE_SCROLL_INSENSITIVE` means that the `ResultSet` can be navigated (scrolled) both forward and backwards. You can also jump to a position relative to the current position, or jump to an absolute position. The `ResultSet` is insensitive to changes in the underlying data source while the `ResultSet` is open. That is, if a record in the `ResultSet` is changed in the database by another thread or process, it will not be reflected in already opened `ResulsSet`'s of this type.
- `TYPE_SCROLL_SENSITIVE` means that the `ResultSet` can be navigated (scrolled) both forward and backwards. You can also jump to a position relative to the current position, or jump to an absolute position. The `ResultSet` is sensitive to changes in the underlying data source while the `ResultSet` is open. That is, if a record in the `ResultSet` is changed in the database by another thread or process, it will be reflected in already opened `ResulsSet`'s of this type.

Method	Description
`absolute()`	Moves the `ResultSet` to point at an absolute position. The position is a row number passed as parameter to the `absolute()` method.
`afterLast()`	Moves the `ResultSet` to point after the last row in the `ResultSet`.
`beforeFirst()`	Moves the `ResultSet` to point before the first row in the `ResultSet`.
`first()`	Moves the `ResultSet` to point at the first row in the `ResultSet`.
`last()`	Moves the `ResultSet` to point at the last row in the `ResultSet`.
`next()`	Moves the `ResultSet` to point at the next row in the `ResultSet`.
`previous()`	Moves the `ResultSet` to point at the previous row in the `ResultSet`.
`relative()`	Moves the `ResultSet` to point to a position relative to its current position. The relative position is passed as a parameter to the relative method, and can be

both positive and negative.

Moves the `ResultSet`

PROGRAM:

```
import java.sql.*;
class A
{
A()
{
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
Statement s=c.createStatement(ResultSet.TPYE_SCROLL_SENSITIVE);
ResultSet r=s.executeQuery("Select *from emp");
While(r.next())
{
String name=r.getString(1);
String usn=r.getString(2);
System.out.println("name="+name);
System.out.println("USN="+usn);
}
r.first();
System.out.println(r.getString(1));
r.last();
System.out.println(r.getString(1));
r.previous();
System.out.println(r.getString(1));
r.absolute(2);
System.out.println(r.getString(1));
r.relative(2);
System.out.println(r.getString(1));
r.relative(-2);
System.out.println(r.getString(1));
c.close();
}
catch(Exception e)
{
S.o.p(e);
}
}
public static void main(String ar[])
{
A a1=new A();
}
}
```

Updatable ResultSet :

- The `ResultSet` concurrency determines whether the `ResultSet` can be updated, or only read.
- A `ResultSet` can have one of two concurrency levels:
    1. ResultSet.CONCUR_READ_ONLY
    2. ResultSet.CONCUR_UPDATABLE
- CONCUR_READ_ONLY means that the `ResultSet` can only be read.
- CONCUR_UPDATABLE means that the `ResultSet` can be both read and updated.
- If a `ResultSet` is updatable, you can update the columns of each row in the `ResultSet`. You do so using the many `updateXXX()` methods.
- `updateRow()` is called that the database is updated with the values of the row

```
import java.sql.*;
class A
{
A()
{
```

```
    import java.sql.*;
    class A
    {
    A()
    {
    try
    {
    Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
    Statement s=c.createStatement(ResultSet.CONCUR_UPDATABLE);
    ResultSet r=s.executeQuery("Select *from emp ");
    r.updateString(1, "Avinash");
    r.updateString(2, "4cb16cs001");

    r.insertRow();
    while(r.next())
    {
    String name=r.getString(1);
    String usn=r.getString(2);
    System.out.println("name="+name);
    System.out.println("USN="+usn);
    }
    c.close();
    }
    catch(Exception e)
    {
    S.o.p(e);
    }
    }
    public stataic void main(String ar[])
    {
```

```
A a1=new A();
}
}
```

Deleteing row from a ResultSet:

- Deleterow() method is nused to delete the row from the databse.
- DeleteRow() method pass as an integer argument ,which specify the row to be deleted.

ResultSet.deleteRow(int);

Program:

```
import java.sql.*;
class A
{
A()
{
Try
{
Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
Statement s=c.createStatement(ResultSet.CONCUR_UPDATABLE);
ResultSet r=s.executeQuery("Select *from emp ");
r.deleteRow(1);
while(r.next())
{
String name=r.getString(1);
String usn=r.getString(2);
System.out.println("name="+name);
System.out.println("USN="+usn);
}
c.close();
```

```
        {
        A()
        {
        try
        {
        Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
        Statement s=c.createStatement();
        c.setAutoCommit(false);
        c.setSavePoint("csb");
        ResultSet r=s.executeQuery("Select *from emp where usn=2");
                r=s.executeQuery("Select *from emp");
        c.releaseSavePoint("csb");
        c.commit();
        }
        c.close();
        }
        catch(Exception e)
        {
        S.o.p(e);
        c.rollback();
        }
        }
        public static void main(String ar[])
        {
        A a1=new A();
        }
        }
```

8.Metadata:

Metadata is data about data. J2ee component can access metadata by using

- DatabaseMetaData interface.
- ResultSetMetaData interface

DatabaseMetaData interface:

- The DatabaseMetaData interface is used to retrieve information about database,table,columns and index amoung other information about dbms.
- J2ee component retrives metadata about the database by calling getMetaData() method of the connection interface object. The getMetaData() method return a DatabaseMetaData object that contain information of database and components.
- Most commonly used DatabaseMetaData interface methods as follows:
    - **getDataBaseProductName()-** returns the product name of the database.
    - **getUserName()-**returns the username of database
    - **getURL()-** returns the URL of the database
    - **getSchemas()-**returns the all schemas of the database which are available
    - **getPrimaryKeys()-**returns the primary key available in the database
    - **getTables()-**returns the table name in the database

program:

```
        import java.sql.*;
        class A
        {
        A()
        {
        try
        {
        Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
        Statement s=c.createStatement();
        ResultSet  r=s.executeQuery("Select *from emp");
        DatabaseMetaData d=c.getMetaData();
        System.out.println(d.getUserNAme());
        System.out.println(d.getTables());
        System.out.println(d.getURL());
        }
        c.close();
        }
        catch(Exception e)
        {
        S.o.p(e);
        }
        }
        public static void main(String ar[])
        {
        A a1=new A();
        }
        }
```

ResultSetMetaData interface:

- ResultSetMetaData interface is used to retrieve the information by calling the getMetaData() method of ResultSet interface.
- Different methods in the ResultSetMetaData inetface are as follows:
    - **getColunmCount()-**returns the number of column available in the table
    - **getColunmName(int)-**returns the name of column specified by the column number
    - **getColunmTye(int**)-returns the type of column specified by the column number

program:

```
import java.sql.*;
class A
{
A()
{
try
{
Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("JDBC:mysql://localhost:3306/CSB","root","");
Statement s=c.createStatement();
ResultSet  r=s.executeQuery("Select *from emp");
ResultSetMetaData d=r.getMetaData();
System.out.println(d.getColunmName(1));
System.out.println(d.getColunmCount());
System.out.println(d.getColunmType(1));
}
c.close();
}
catch(Exception e)
{
S.o.p(e);
}
}
public static void main(String ar[])
{
A a1=new A();
}
}
```

Data Types:

The JDBC driver converts the Java data type to the appropriate JDBC type, before sending it to the database. It uses a default mapping for most data types. The following table summarizes the default JDBC data type that the Java data type is converted to, when you call the setXXX() method.

SQL	JDBC/Java
VARCHAR	String
CHAR	String
LONGVARCHAR	String
BIT	boolean
NUMERIC	java.math.BigDecimal
TINYINT	byte
SMALLINT	short