

EXAMPLE -1

```
import java.io.*;
public class Student {

    int sage;

    public Student(String name) {
        // This constructor has one parameter, name.
        System.out.println("Name of student is :" + name );
    }

    public void setAge( int age ) {
        sage = age;
    }

    public int getAge( ) {
        System.out.println("Student's age is :" + sage );
        return sage;
    }

    public static void main(String []args) {

        /* Object creation */
        Student S = new Student( "XYZ" );

        /* Call class method to set age */
        S.setAge( 2 );

        /* Call another class method to get age */
        S.getAge( );

        /* You can access instance variable as follows as well */
        System.out.println("Variable Value :" + S.sage );
    }
}
```

EXAMPLE -2

```
import java.io.*;
public class Employee{

    String name;
    int age;
    String designation;
    double salary;

    // This is the constructor of the class Employee
    public Employee(String name){
        this.name = name;
    }

    // Assign the age of the Employee to the variable age.
    public void empAge(int empAge){
        age = empAge;
    }

    /* Assign the designation to the variable designation.*/
    public void empDesignation(String empDesig){
        designation = empDesig;
    }

    /* Assign the salary to the variable salary.*/
    public void empSalary(double empSalary){
        salary = empSalary;
    }

    /* Print the Employee details */
    public void printEmployee(){
        System.out.println("Name:"+ name );
        System.out.println("Age:"+ age );
        System.out.println("Designation:"+ designation );
        System.out.println("Salary:"+ salary);
    }
}
```

```
import java.io.*;
public class EmployeeTest{

    public static void main(String args[]){

        /* Create two objects using constructor */
        Employee empOne = new Employee("James Smith");
        Employee empTwo = new Employee("Mary Anne");

        // Invoking methods for each object created
        empOne.empAge(26);
        empOne.empDesignation("Senior Software Engineer");
        empOne.empSalary(1000);
        empOne.printEmployee();

        empTwo.empAge(21);
        empTwo.empDesignation("Software Engineer");
        empTwo.empSalary(500);
        empTwo.printEmployee();
    }
}
```

Access Modifiers

- Access modifiers are used to control the accessibility to class, constructor, variable, method or data member.
- In other words, we can use access modifiers to protect data and behaviors from the outside world.
- There are four types of access modifiers available in java:
 - Public
 - Protected
 - Default – No keyword required
 - Private
- The order of the access modifiers from the least restrictive to the most restrictive:
 - **public > protected > default > private**

Java public access modifier:

- When applied to a class, the class is accessible from any classes regardless of packages.
- This is the least restrictive access modifier which means the widest range of accessibility, or visibility.
- When applied to a member, the member is accessible from any classes.

EXAMPLE -3

```
package p1;

public class A {

    public void display() {
        System.out.println("Hello World");
    }
}
```

```

package p2;

import p1.*;

public class B {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        {
            A obj = new A();
            obj.display();
        }
    }
}

```

Output: Hello World

Java protected access modifier:

- The protected access modifier is specified using the keyword **protected**.
- This is more restrictive than the public modifier. It is applied for members only.
 - There is no ‘protected’ class.
- When a member of a class is declared as protected,
 - It is accessible by only classes in the same package or by a subclass in different package.

EXAMPLE - 4

```

package p1;

public class A {

    protected void display() {
        System.out.println("Hello World");
    }
}

```

```

        }
    }

package p2;

import p1.*;
public class B {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        A obj = new A();
        obj.display();
    }
}

```

Output: Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method display() from the type A is not visible
at p2.B.main([B.java:9](#))

EXAMPLE - 5

```

package p1;

public class A {

    protected void display() {
        System.out.println("Hello World");
    }
}

```

```

package p2;

import p1.*;
public class B {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
    }
}

```

```
B obj = new B();
obj.display();
}

}
```

Output: Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method display() is undefined for the type B
at p2.B.main(B.java:9)

EXAMPLE - 6

```
package p1;

public class A {

    protected void display() {
        System.out.println("Hello World");
    }
}
```

```
package p2;

import p1.*;

public class B extends A      //Class B is subclass of A
{
    public static void main(String[] args) {

        B obj = new B();
        obj.display();
    }
}
```

Output: Hello World

Java default access modifier:

- When no access modifier is specified for a class, method or data member
 - It is said to be having the **default** access modifier by default.
- It is more restrictive than the **protected** modifier.
- The data members, class or methods which are not declared using any access modifiers i.e. having default access modifier are accessible **only within the same package**.

EXAMPLE - 7

```
package p1;

public class A {

    void display() {
        System.out.println("Hello World");
    }
}
```

```
package p2;
import p1.*;
public class B {
    public static void main(String[] args) {

        //accessing class A from package p1
        A obj = new A();
        obj.display();
    }
}
```

Output: Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method display() from the type A is not visible
at p2.B.main([B.java:9](#))

EXAMPLE - 8

```
package p1;

public class A {

    void display() {
        System.out.println("Hello World");
    }
}

package p1;

public class B {

    public static void main(String[] args) {

        A obj = new A();
        obj.display();
    }
}
```

Output: Hello World

Java private access modifier:

- The private access modifier is specified using the keyword **private**.
- This is the most restrictive access modifier in Java.
- The methods or data members declared as private are accessible only within the class in which they are declared.
- Any other **class of same package will not be able to access** these members.

EXAMPLE - 9

```
package p1;

public class A {

    private void display()
    {
        System.out.println("Hello World");
    }
}

public class B {

    public static void main(String[] args) {

        //trying to access private method of another class
        A obj = new A();
        obj.display();
    }
}
```

Output: Exception in thread "main" java.lang.Error: Unresolved compilation problem:

The method display() from the type A is not visible
at p1.B.main([A.java:16](#))

EXAMPLE – 10

```
package p1;

public class A {

    private void display()
    {
        System.out.println("Hello World");
    }
}
```

```
public static void main(String[] args) {  
    A obj = new A();  
    obj.display();  
}  
}
```

Output: Hello World

Java static keyword

- The **static keyword** in Java is used for memory management mainly.
- java static keyword can be applied to :
 - Variable (also known as a class variable)
 - Method (also known as a class method)
 - Block
 - Nested class

1) Java static variable

- If you declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- It makes your program **memory efficient** (i.e., it saves memory).
- The static variable gets memory only once in the class area at the time of class loading.

EXAMPLE – 11: Java Program to demonstrate the use of static variable .

```
class Student{  
    int rollno;                                //instance variable  
    String name;  
    static String college ="ITS";                //static variable  
  
    //constructor  
    Student(int r, String n){  
        rollno = r;  
        name = n;  
    }  
  
    //method to display the values  
    void display (){  
        System.out.println(rollno+" "+name+" "+college);  
    }  
}  
  
//Test class to show the values of objects  
public class TestStaticVariable1{  
  
    public static void main(String args[]){  
  
        Student s1 = new Student(111,"Karan");  
        Student s2 = new Student(222,"Aryan");  
  
        //we can change the college of all objects by the single line  
        //Student.college="BBDIT";  
        s1.display();  
        s2.display();  
    }  
}
```

Output:

111 Karan ITS
222 Aryan ITS

2) Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

EXAMPLE – 12

```
//Java Program to demonstrate the use of a static method.

class Student{
    int rollno;
    String name;
    static String college = "ITS";

    //static method to change the value of static variable
    static void change(){
        college = "BBDIT";
    }

    //constructor to initialize the variable
    Student(int r, String n){
        rollno = r;
        name = n;
    }

    //method to display values
    void display(){
        System.out.println(rollno+" "+name+" "+college);
    }
}
```

```

//Test class to create and display the values of object
public class TestStaticMethod{

    public static void main(String args[]){

        Student.change();//calling change method

        //creating objects
        Student s1 = new Student(111,"Karan");
        Student s2 = new Student(222,"Aryan");
        Student s3 = new Student(333,"Sonoo");

        //calling display method
        s1.display();
        s2.display();
        s3.display();
    }
}

```

Output:

```

111 Karan BBDIT
222 Aryan BBDIT
333 Sonoo BBDIT

```

NOTE: Why is the Java main method static?

- It is because the object is not required to call a static method.
- If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

3) Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of class loading.

EXAMPLE – 13

```
class A2{  
  
    static{System.out.println("static block is invoked");}  
  
    public static void main(String args[]){  
        System.out.println("Hello main");  
    }  
}
```

Output:

static block is invoked
Hello main

4) Java static nested class

- A static class i.e. created inside a class is called static nested class in java.
- It cannot access non-static data members and methods. It can be accessed by outer class name.
- It can access static data members of outer class including private.
- Static nested class cannot access non-static (instance) data member or method.

EXAMPLE – 14

```
class TestOuter1 {  
  
    static int data=30;  
  
    static class Inner{  
  
        void msg(){  
            System.out.println("data is "+data);  
        }  
    }  
}
```

```
public static void main(String args[]){
    TestOuter1.Inner obj=new TestOuter1.Inner();
    obj.msg();
}
}
```

Output:

data is 30

How to get the value of Environment variables?

- The [System class in Java](#) provides a method named **System.getenv()**
- The **System.getenv()** is used to get the value of an environment variable set in the current system.
- This method returns the string value of the variable, or null if the variable is not defined in the system environment.
- Syntax:
 - **System.getenv(name)** // name is the environment variable name.

Example: 1

```
public class SysEnv {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        // Get the value of the TEMP environment variable  
        System.out.print("System.getenv(\"TEMP\") = ");  
        System.out.println(System.getenv("TEMP"));  
  
        // Get the value of the OS environment variable  
        System.out.print("System.getenv(\"OS\") = ");  
        System.out.println(System.getenv("OS"));  
  
        // Get the value of the JAVA_HOME environment variable  
        System.out.print("System.getenv(\"JAVA_HOME\") = ");  
        System.out.println(System.getenv("JAVA_HOME"));  
  
        // gets the value of the specified environment variable "PATH"  
        System.out.print("System.getenv(\"PATH\") = ");  
        System.out.println(System.getenv("PATH"));  
  
        // gets the value of the specified environment variable "TEMP"  
        System.out.print("System.getenv(\"TEMP\") = ");
```

```

        System.out.println(System.getenv("TEMP"));

        // gets the value of the specified environment variable "USERNAME"
        System.out.print("System.getenv(\"USERNAME\") = ");
        System.out.println(System.getenv("USERNAME"));
    }

}

```

Output:

```

System.getenv("TEMP") =C:\Users\Sangram\AppData\Local\Temp
System.getenv("OS") =Windows_NT
System.getenv("JAVA_HOME") = null
System.getenv("PATH") = C:/Program
Files/Java/jdk8/bin/..../jre/bin/server;C:/Program
Files/Java/jdk8/bin/..../jre/bin;C:/Program
Files/Java/jdk8/bin/..../jre/lib/amd64;C:\Program
Files\Dell\DW WLANcard;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDO
WS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Pro
gramFiles\HDF_Group\HDFView\2.10.1\bin;C:\ProgramFiles\MATLAB\R2014a\
runtime\win64;C:\ProgramFiles\MATLAB\R2014a\bin;C:\ProgramFiles\MATLA
B\R2014a\polyspace\bin;C:\ProgramFiles(x86)\MATLAB\R2014a\runtime\win32;
C:\Program Files (x86)\MATLAB\R2014a\bin;C:\Program Files
(x86)\MATLAB\R2014a\polyspace\bin;C:\ProgramFiles\MiKTeX2.9\miktex\bin\x
64\;C:\ProgramFiles(x86)\Skype\Phone\;C:\WINDOWS\System32\OpenSSH\;C:\
ProgramFiles\Java\jdk8\bin;C:\ProgramFiles\Java\jre8\bin;C:\Users\Sangram\App
Data\Local\Microsoft\WindowsApps;;C:\eclipse;
System.getenv("TEMP") = C:\Users\Sangram\AppData\Local\Temp
System.getenv("USERNAME") = Sangram

```

Example 2: Java program to get the value of all environment variables at once using System.getenv() method

```
import java.util.Map;

public class SysEnvAll {

    public static void main(String[] args) {

        // Get the value of all environment variables at once
        // and store it in Map
        Map<String, String> env = System.getenv();

        for (String envName : env.keySet()) {

            System.out.format("%s=%s%n", envName, env.get(envName));
        }
    }
}
```

Output:

```
USERDOMAIN_ROAMINGPROFILE=Sangram-PC
LOCALAPPDATA=C:\Users\Sangram\AppData\Local
PROCESSOR_LEVEL=6
FP_NO_HOST_CHECK=NO
USERDOMAIN=Sangram-PC
FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
LOGONSERVER=\\SANGRAM-PC
SESSIONNAME=Console
ALLUSERSPROFILE=C:\ProgramData
PROCESSOR_ARCHITECTURE=AMD64
PSModulePath=C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules\
SystemDrive=C:
OneDrive=C:\Users\Sangram\OneDrive
APPDATA=C:\Users\Sangram\AppData\Roaming
USERNAME=Sangram
windows_tracing_logfile=C:\BVTBin\Tests\installpackage\csilogfile.log
ProgramFiles(x86)=C:\Program Files (x86)
CommonProgramFiles=C:\Program Files\Common Files
```

Path=C:/Program Files/Java/jdk8/bin/../jre/bin/server;C:/Program
Files/Java/jdk8/bin/../jre/bin;C:/Program Files/Java/jdk8/bin/../jre/lib/amd64;C:\Program
Files\Del\DW WLAN
Card;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WIN
DOWS\System32\WindowsPowerShell\v1.0\;C:\Program
Files\HDF_Group\HDFView\2.10.1\bin;C:\Program
Files\MATLAB\R2014a\runtime\win64;C:\Program
Files\MATLAB\R2014a\bin;C:\Program
Files\MATLAB\R2014a\polyspace\bin;C:\Program Files
(x86)\MATLAB\R2014a\runtime\win32;C:\Program Files
(x86)\MATLAB\R2014a\bin;C:\Program Files
(x86)\MATLAB\R2014a\polyspace\bin;C:\Program Files\MiKTeX
2.9\miktex\bin\x64;C:\Program Files
(x86)\Skype\Phone\;C:\WINDOWS\System32\OpenSSH\;C:\Program
Files\Java\jdk8\bin;C:\Program
Files\Java\jre8\bin;C:\Users\Sangram\AppData\Local\Microsoft\WindowsApps;;C:\eclips
e;
FPS_BROWSER_USER_PROFILE_STRING=Default
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
DriverData=C:\Windows\System32\Drivers\DriverData
OS=Windows_NT
windows_tracing_flags=3
COMPUTERNAME=SANGRAM-PC
PROCESSOR_REVISION=2a07
CommonProgramW6432=C:\Program Files\Common Files
ComSpec=C:\WINDOWS\system32\cmd.exe
ProgramData=C:\ProgramData
ProgramW6432=C:\Program Files
HOMEPATH=\Users\Sangram
SystemRoot=C:\WINDOWS
TEMP=C:\Users\Sangram\AppData\Local\Temp
HOMEDRIVE=C:
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 42 Stepping 7, GenuineIntel
USERPROFILE=C:\Users\Sangram
TMP=C:\Users\Sangram\AppData\Local\Temp
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
ProgramFiles=C:\Program Files
PUBLIC=C:\Users\Public
NUMBER_OF_PROCESSORS=4
windir=C:\WINDOWS
=::=::\

How to get the value of System Property?

- The `System.getProperty(key)` method gets the system property indicated by the specified key.
- key is the name of the system property.
- This method returns the string value of the system property, or null if there is no property with that key.

Example 3:

```
public class SysProp {  
  
    public static void main(String[] args) {  
  
        // Printing Name of the system property  
        System.out.print("user.dir: ");  
        System.out.println(System.getProperty("user.dir"));  
  
        // Fetches the property set with 'home' key  
        System.out.print("home: ");  
        System.out.println(System.getProperty("home"));  
  
        // Printing 'name of Operating System'  
        System.out.print("os.name: ");  
        System.out.println(System.getProperty("os.name"));  
  
        // Printing 'Version of Operating System'  
        System.out.print("os.version: ");  
        System.out.println(System.getProperty("os.version"));  
  
        // Printing 'File Separator'  
        System.out.print("file.separator: ");  
        System.out.println(System.getProperty("file.separator"));  
  
        // Printing 'JAVA Runtime version'  
        System.out.print("JDK version: ");  
        System.out.println(System.getProperty("java.runtime.version"));  
    }  
}
```

```

// Printing 'JAVA Class Path'
System.out.print("java.class.path: ");
System.out.println(System.getProperty("java.class.path"));

// Printing 'name' property
System.out.print("name: ");
System.out.println(System.getProperty("name"));
}

}

```

Output:

user.dir: C:\Users\Sangram\eclipse-workspace\SysProp
home: null
os.name: Windows 10
os.version: 10.0
file.separator: \
version: 1.8.0_171-b11
java.class.path: C:\Program Files\Java\jdk8\jre\lib\resources.jar;C:\Program Files\Java\jdk8\jre\lib\rt.jar;C:\Program Files\Java\jdk8\jre\lib\jsse.jar;C:\Program Files\Java\jdk8\jre\lib\jce.jar;C:\Program Files\Java\jdk8\jre\lib\charsets.jar;C:\Program Files\Java\jdk8\jre\lib\ext\access-bridge-64.jar;C:\Program Files\Java\jdk8\jre\lib\ext\cldrdata.jar;C:\Program Files\Java\jdk8\jre\lib\ext\dnsns.jar;C:\Program Files\Java\jdk8\jre\lib\ext\jaccess.jar;C:\Program Files\Java\jdk8\jre\lib\ext\jfxrt.jar;C:\Program Files\Java\jdk8\jre\lib\ext\localizedata.jar;C:\Program Files\Java\jdk8\jre\lib\ext\nashorn.jar;C:\Program Files\Java\jdk8\jre\lib\ext\sunec.jar;C:\Program Files\Java\jdk8\jre\lib\ext\sunjce_provider.jar;C:\Program Files\Java\jdk8\jre\lib\ext\sunmscapi.jar;C:\Program Files\Java\jdk8\jre\lib\ext\sunpkcs11.jar;C:\Program Files\Java\jdk8\jre\lib\ext\zipfs.jar;C:\Users\Sangram\eclipse-workspace\SysProp\bin
name: null

How to get all the value of System Property?

- The [**System.getProperties\(\)**](#): fetches all the properties – values that the JVM on your System gets from the Operating System.

Example: 4

```
public class SysPropAll {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        // this will list the current system properties  
        System.out.println(System.getProperties());  
    }  
  
}
```

Output:

```
{java.runtime.name=Java(TM) SE Runtime Environment,  
sun.boot.library.path=C:\Program Files\Java\jdk8\jre\bin, java.vm.version=25.171-  
b11, java.vm.vendor=Oracle Corporation, java.vendor.url=http://java.oracle.com/,  
path.separator=;, java.vm.name=Java HotSpot(TM) 64-Bit Server VM,  
file.encoding.pkg=sun.io, user.country=US, user.script=,  
sun.java.launcher=SUN_STANDARD, sun.os.patch.level=,  
java.vm.specification.name=Java Virtual Machine Specification,  
user.dir=C:\Users\Sangram\eclipse-workspace\SysPropAll,  
java.runtime.version=1.8.0_171-b11,  
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment,  
java.endorsed.dirs=C:\Program Files\Java\jdk8\jre\lib\endorsed, os.arch=amd64,  
java.io.tmpdir=C:\Users\Sangram\AppData\Local\Temp\, line.separator=,  
, java.vm.specification.vendor=Oracle Corporation, user.variant=,  
os.name=Windows 10, sun.jnu.encoding=Cp1252, java.library.path=C:\Program  
Files\Java\jdk8\bin;C:\WINDOWS\Sun\Java\bin;C:\WINDOWS\system32;C:\WI  
NDOWS;C:/Program Files/Java/jdk8/bin/..../jre/bin/server;C:/Program  
Files/Java/jdk8/bin/..../jre/bin;C:/Program  
Files/Java/jdk8/bin/..../jre/lib/amd64;C:\Program Files\Dell\DW WLAN
```

Card;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;
C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\Program
Files\HDF_Group\HDFView\2.10.1\bin;C:\Program
Files\MATLAB\R2014a\runtime\win64;C:\Program
Files\MATLAB\R2014a\bin;C:\Program
Files\MATLAB\R2014a\polyspace\bin;C:\Program Files
(x86)\MATLAB\R2014a\runtime\win32;C:\Program Files
(x86)\MATLAB\R2014a\bin;C:\Program Files
(x86)\MATLAB\R2014a\polyspace\bin;C:\Program Files\MiKTeX
2.9\miktex\bin\x64\;C:\Program Files
(x86)\Skype\Phone\;C:\WINDOWS\System32\OpenSSH\;C:\Program
Files\Java\jdk8\bin;C:\Program
Files\Java\jre8\bin;C:\Users\Sangram\AppData\Local\Microsoft\WindowsApps;;C:
\eclipse\;;, java.specification.name=Java Platform API Specification,
java.class.version=52.0, sun.management.compiler=HotSpot 64-Bit Tiered
Compilers, os.version=10.0, user.home=C:\Users\Sangram, user.timezone=,
java.awt.printerjob=sun.awt.windows.WPrinterJob, file.encoding=Cp1252,
java.specification.version=1.8, java.class.path=C:\Program
Files\Java\jdk8\jre\lib\resources.jar;C:\Program
Files\Java\jdk8\jre\lib\rt.jar;C:\Program Files\Java\jdk8\jre\lib\jsse.jar;C:\Program
Files\Java\jdk8\jre\lib\jce.jar;C:\Program
Files\Java\jdk8\jre\lib\charsets.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\access-bridge-
64.jar;C:\Program Files\Java\jdk8\jre\lib\ext\cldrdata.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\dnsns.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\jaccess.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\jfxrt.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\.localedata.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\nashorn.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\sunec.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\sunjce_provider.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\sunmscapi.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\sunpkcs11.jar;C:\Program
Files\Java\jdk8\jre\lib\ext\zipfs.jar;C:\Users\Sangram\eclipse-
workspace\SysPropAll\bin, user.name=Sangram,
java.vm.specification.version=1.8, sun.java.command=SysPropAll,
java.home=C:\Program Files\Java\jdk8\jre, sun.arch.data.model=64,
user.language=en, java.specification.vendor=Oracle Corporation,
awt.toolkit=sun.awt.windows.WToolkit, java.vm.info=mixed mode,
java.version=1.8.0_171, java.ext.dirs=C:\Program

```
Files\Java\jdk8\jre\lib\ext;C:\WINDOWS\Sun\Java\lib\ext,
sun.boot.class.path=C:\Program Files\Java\jdk8\jre\lib\resources.jar;C:\Program
Files\Java\jdk8\jre\lib\rt.jar;C:\Program
Files\Java\jdk8\jre\lib\sunrsasign.jar;C:\Program
Files\Java\jdk8\jre\lib\jsse.jar;C:\Program Files\Java\jdk8\jre\lib\jce.jar;C:\Program
Files\Java\jdk8\jre\lib\charsets.jar;C:\Program
Files\Java\jdk8\jre\lib\jfr.jar;C:\Program Files\Java\jdk8\jre\classes,
java.vendor=Oracle Corporation, file.separator=\,
java.vendor.url.bug=http://bugreport.sun.com/bugreport/,
sun.io.unicode.encoding=UnicodeLittle, sun.cpu.endian=little,
sun.desktop=windows, sun.cpu.isalist=amd64}
```

Example: 5 Java Program illustrating the working of `getProperties()` method.

```
public class SysPropAll {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        // this will list the current system properties
        java.util.Properties jvm = System.getProperties();
        jvm.list(System.out);
    }
}
```

Note: System class refers to the JVM on which you are compiling your JAVA code `getProperty` fetches the actual properties that JVM on your System gets from your Operating System.

Output:

```
-- listing properties --
java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path=C:\Program Files\Java\jdk8\jre\bin
java.vm.version=25.171-b11
java.vm.vendor=Oracle Corporation
java.vendor.url=http://java.oracle.com/
path.separator=;
```

java.vm.name=Java HotSpot(TM) 64-Bit Server VM
file.encoding.pkg=sun.io
user.script=
user.country=US
sun.java.launcher=SUN_STANDARD
sun.os.patch.level=
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\Users\Sangram\eclipse-workspace\Sy...
java.runtime.version=1.8.0_171-b11
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:\Program Files\Java\jdk8\jre\lib\en...
os.arch=amd64
java.io.tmpdir=C:\Users\Sangram\AppData\Local\Temp\
line.separator=

java.vm.specification.vendor=Oracle Corporation
user.variant=
os.name=Windows 10
sun.jnu.encoding=Cp1252
java.library.path=C:\Program Files\Java\jdk8\bin;C:\WIN...
java.specification.name=Java Platform API Specification
java.class.version=52.0
sun.management.compiler=HotSpot 64-Bit Tiered Compilers
os.version=10.0
user.home=C:\Users\Sangram
user.timezone=
java.awt.printerjob=sun.awt.windows.WPrinterJob
file.encoding=Cp1252
java.specification.version=1.8
user.name=Sangram
java.class.path=C:\Program Files\Java\jdk8\jre\lib\re...
java.vm.specification.version=1.8
sun.arch.data.model=64
java.home=C:\Program Files\Java\jdk8\jre
sun.java.command=SysPropAll
java.specification.vendor=Oracle Corporation
user.language=en
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode
java.version=1.8.0_171

```
java.ext.dirs=C:\Program Files\Java\jdk8\jre\lib\ex...
sun.boot.class.path=C:\Program Files\Java\jdk8\jre\lib\re...
java.vendor=Oracle Corporation
file.separator=\
java.vendor.url.bug=http://bugreport.sun.com/bugreport/
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
sun.desktop=windows
sun.cpu.isalist=amd64
```

Example: 6

```
import java.io.IOException;
public class SysPropAll {

    public static void main(String[] argv) throws IOException {

        if (argv.length == 0)
            System.getProperties().list(System.out);
        else {
            for (String s : argv) {
                System.out.println(s + " = " + System.getProperty(s));
            }
        }
    }
}
```

Output:

```
java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path=C:\Program Files\Java\jdk8\jre\bin
java.vm.version=25.171-b11
java.vm.vendor=Oracle Corporation
java.vendor.url=http://java.oracle.com/
path.separator=;
java.vm.name=Java HotSpot(TM) 64-Bit Server VM
file.encoding.pkg=sun.io
user.script=
```

user.country=US
sun.java.launcher=SUN_STANDARD
sun.os.patch.level=
java.vm.specification.name=Java Virtual Machine Specification
user.dir=C:\Users\Sangram\eclipse-workspace\Sy...
java.runtime.version=1.8.0_171-b11
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:\Program Files\Java\jdk8\jre\lib\en...
os.arch=amd64
java.io.tmpdir=C:\Users\Sangram\AppData\Local\Temp\
line.separator=

java.vm.specification.vendor=Oracle Corporation
user.variant=
os.name=Windows 10
sun.jnu.encoding= Cp1252
java.library.path=C:\Program Files\Java\jdk8\bin;C:\WIN...
java.specification.name=Java Platform API Specification
java.class.version=52.0
sun.management.compiler=HotSpot 64-Bit Tiered Compilers
os.version=10.0
user.home=C:\Users\Sangram
user.timezone=
java.awt.printerjob=sun.awt.windows.WPrinterJob
file.encoding= Cp1252
java.specification.version=1.8
user.name=Sangram
java.class.path=C:\Program Files\Java\jdk8\jre\lib\re...
java.vm.specification.version=1.8
sun.arch.data.model=64
java.home=C:\Program Files\Java\jdk8\jre
sun.java.command=SysPropAll
java.specification.vendor=Oracle Corporation
user.language=en
awt.toolkit=sun.awt.windows.WToolkit
java.vm.info=mixed mode
java.version=1.8.0_171
java.ext.dirs=C:\Program Files\Java\jdk8\jre\lib\ex...
sun.boot.class.path=C:\Program Files\Java\jdk8\jre\lib\re...
java.vendor=Oracle Corporation

```
file.separator=\
java.vendor.url.bug=http://bugreport.sun.com/bugreport/
sun.cpu.endian=little
sun.io.unicode.encoding=UnicodeLittle
sun.desktop=windows
sun.cpu.isalist=amd64
```

Test the presence of a particular class

Example: 7 Check the Java Swing is Present or not.

```
public class CheckForSwing {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        try {
            Class.forName("javax.swing.JButton");
            System.out.println("Java Swing is present");
        } catch (ClassNotFoundException e) {

            String failure = "Sorry, but this version of MyApp needs \n" +
                "a Java Runtime with JFC/Swing components\n" +
                "having the final names (javax.swing.*)";

            System.err.println(failure);
        }
    }
}
```

Output:

Java Swing is present

Print Null device of your System

- In some operating systems, the null device is a device file that discards all data written to it but reports that the write operation succeeded.
- It provides no data to any process that reads from it, yielding EOF immediately.
- In Unix and Linux System, /dev/null
- In Windows System, NUL:
- Other Systems, jnk

Example: 8

```
import java.io.File;

public class TestNullDevice {

    final static String UNIX_NULL_DEV = "/dev/null";
    final static String WINDOWS_NULL_DEV = "NUL:";
    final static String FAKE_NULL_DEV = "jnk";

    public static String getNullDev(){

        if(new File(UNIX_NULL_DEV).exists()){
            return UNIX_NULL_DEV;
        }

        String sys=System.getProperty("os.name");

        if(sys==null){
            return FAKE_NULL_DEV;
        }

        if(sys.startsWith("Windows")){
            return WINDOWS_NULL_DEV;
        }
    }
}
```

```
    }

    return FAKE_NULL_DEV;
}

/** Return the name of the "Null Device" on platforms which support it,
* or "jnk" (to create an obviously well-named temp file) otherwise.*/

public static void main(String[] args) {
    // TODO Auto-generated method stub
    String devNull=getNullDev();
    System.out.println(devNull);
}

}
```

Output:

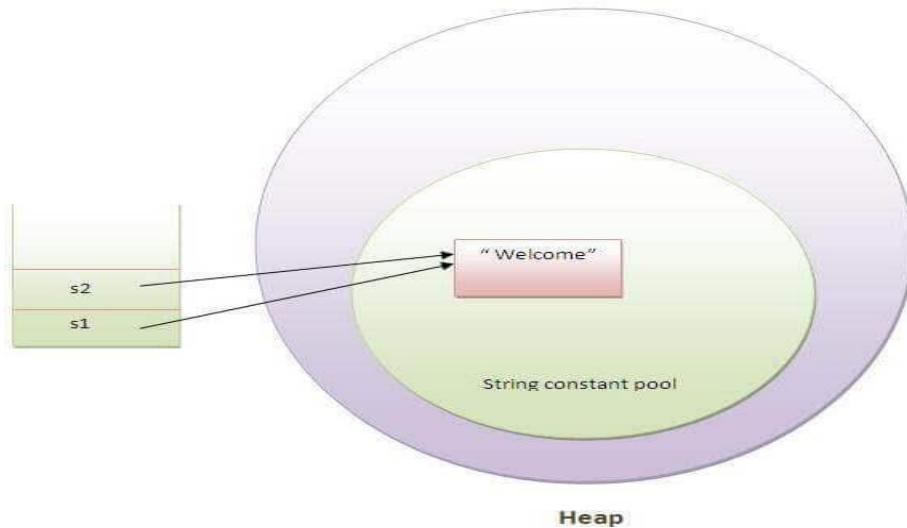
NUL:

Java String

- In Java, the string is basically an object that represents a sequence of char values. An array of characters works same as Java string.
- To create strings in java by using these three classes
 - String class
 - StringBuffer class
 - StringBuilder class
- The Java String is immutable, which means it cannot be changed.
 - Whenever we change any string, a new instance is created.
 - For mutable strings, you can use StringBuffer and StringBuilder classes.
- **String class**
 - The *java.lang.String* class is used to create a string object.
 - There are two ways to create String object:
 - string literal
 - new keyword
- **String Literal**
 - `String s1="Welcome";` *//the JVM checks the "string constant pool" first. If the string doesn't exist in the pool, a new string instance is created and placed in the pool.*
 - `String s2="Welcome"` *//It doesn't create a new instance because the string already exists in the*

pool, a reference to the pooled instance is returned.

- String objects are stored in a special memory area known as the "string constant pool".



- **New Keyword**

- `String s=new String("Welcome");`

//creates two objects and one reference variable.
JVM will create a new string object in normal
(non-pool) heap memory, and the literal
"Welcome" will be placed in the string constant
pool. The variable 's' will refer to the object in a
heap (non-pool).

Example – 1: Create String by using String literal and new keyword.

```
public class StringExample{  
  
    public static void main(String args[]){  
  
        //creating string by java string literal  
        String s1="java";  
  
        char ch[]={ 's','t','r','i','n','g','s'}; //character array
```

```

//converting char array to string
String s2=new String(ch);

//creating java string by new keyword
String s3=new String("example");

//output
System.out.println("1st String : " + s1);
System.out.println("2nd String : " + s2);
System.out.println("3rd String : " + s3);

}

}

```

Output:

```

1st String : java
2nd String : strings
3rd String : example

```

String Length

The length of a string can be found by the **length()** method.

Example: Find the length of the String.

```

public class Length {

    public static void main(String[] args) {

        String txt = "ABCDEFGHIJKLMNPQRSTUVWXYZ";
        System.out.println("The length of the text string is: " + txt.length());
    }
}

```

Output:

The length of the text string is: 26

To Upper and To Lower

Make the string to upper or lower case.

Example:

```
public class UpperLower {  
  
    public static void main(String[] args) {  
  
        String txt = "Hello World";  
  
        System.out.println("The upper case is " + txt.toUpperCase());  
  
        System.out.println("The lower case is " + txt.toLowerCase());  
    }  
}
```

Output:

The upper case is HELLO WORLD

The lower case is hello world

Finding Index of a Character in a String

- The `indexOf()` method returns the **index** (the position) of the first occurrence of a specified text in a string (including whitespace).

Example: Find the index of "locate" in a string "Please locate where 'locate' occurs!", by using `indexOf()` method.

```
public class FindCharIndex {  
  
    public static void main(String[] args) {  
  
        String txt = "Please locate where 'locate' occurs!";  
        int i=txt.indexOf("locate");  
        //int i=txt.indexOf('e');
```

```
        System.out.println("The index value is " + i);
    }
}
```

Output:

The index value is 7

Processing a String One Character at a Time

- A string's **charAt()** method retrieves a given character by index number (starting at zero) from within the String object.
- To process all the characters in a String, one after another, use a for loop ranging from zero to `String.length()-1`.

Example : Retrieves a given character by index number

```
public class CharAtExample{

    public static void main(String args[]){

        String name="Hello";

        char ch=name.charAt(4); /returns the char value at the 4th index

        System.out.println("Character at index 4 is "+ ch);

    }

}
```

Output:

Character at index 4 is o

Example: Process all the characters in a String:

```
public class StrCharAt {

    public static void main(String[] av) {
```

```
String a = "A quick bronze fox kept a lazy bovine";  
  
for (int i=0; i < a.length(); i++) //Don't use for each  
  
    System.out.println("Char " + i + " is " + a.charAt(i));  
  
}  
}
```

Output:

Char 0 is A
Char 1 is
Char 2 is q
Char 3 is u
Char 4 is i
Char 5 is c
Char 6 is k
Char 7 is
Char 8 is b
Char 9 is r
Char 10 is o
Char 11 is n
Char 12 is z
Char 13 is e
Char 14 is
Char 15 is f
Char 16 is o
Char 17 is x
Char 18 is
Char 19 is l
Char 20 is e
Char 21 is p
Char 22 is t
Char 23 is
Char 24 is a
Char 25 is
Char 26 is l
Char 27 is a

```
Char 28 is z
Char 29 is y
Char 30 is
Char 31 is b
Char 32 is o
Char 33 is v
Char 34 is i
Char 35 is n
Char 36 is e
```

Example: process all the characters in a String using toCharArray() method.

```
public class ForEachChar {

    public static void main(String[] args) {

        String str = "Hello world";

        // toCharArray() returns an Array of chars after
        // converting a String into sequence of characters.
        char[] array= str.toCharArray();

        //for (char ch : s) {...} Does not work, in Java 7

        for (char ch : array) {

            System.out.println(ch);
        }
    }
}
```

Output:

```
H
e
l
l
o
```

```
w
```

o
r
l
d

Example: A “checksum” is a numeric quantity representing and confirming the contents of a file. Write a program which reads a string from the user and find its checksum.

```
import java.util.*;  
import java.util.Scanner;  
  
public class StringDemo {  
  
    public static void main(String[] args) {  
  
        Scanner S = new Scanner(System.in);  
        System.out.println("Enter the 1st string ");  
        String s1 = S.nextLine();  
        System.out.println("The 1st string: " + s1);  
  
        int i, j, sum = 0;  
        for (i=0; i<s1.length(); i++) {  
            j= s1.charAt(i);  
            System.out.println("The value of " + s1.charAt(i) + " is " + j);  
            sum += s1.charAt(i);  
        }  
        System.out.println("The checksum is " + sum);  
    }  
}
```

Output:

```
Enter the 1st string  
abc  
The 1st string: abc  
The value of a is 97  
The value of b is 98  
The value of c is 99  
The checksum is 294
```

Taking Strings Apart with Substrings

- **substring()** method returns a part of the string.
- There are two types of substring methods in java string.
 - *substring(int startIndex)*
 - return all the characters from a given index (startIndex) to end of the string.
 - *substring(int startIndex, int endIndex)*
 - return all the characters between the starting index (startIndex) to ending index (endIndex) of the string.

Example:

```
public class SubStringDemo {  
  
    public static void main(String[] av) {  
  
        String a = "Java is great.";  
        System.out.println(a);  
  
        String b = a.substring(5); // b is the String "is great."  
        System.out.println("The b substring is " + b);  
  
        String c = a.substring(5,7); // c is the String "is"  
        System.out.println("The c substring is " + c);  
  
        String d = a.substring(5,a.length()); // d is "is great."  
        System.out.println("The d substring is " + d);  
    }  
}
```

Output:

Java is great.
The b substring is is great.
The c substring is is

The d substring is is great.

Example: Write a program which reads two strings from the user and concat the substring from 0 to 8 index of the first string with the 0 to 5 indexed of the second string and print the resultant string. The two strings are as follows:

ITER is my College.
ITER is in SOA.

```
import java.util.*;  
import java.util.Scanner;  
  
public class StringDemo {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        Scanner S = new Scanner(System.in);  
  
        System.out.println("Enter the 1st string ");  
        String s1 = S.nextLine();  
        System.out.println("The 1st string: " + s1);  
  
        System.out.println("Enter the 2nd string ");  
        String s2 = S.nextLine();  
        System.out.println("The 2nd string: " + s2);  
  
        String a = s1.substring(0,8);  
        System.out.println("The 1st substring is: " + a);  
  
        String b = s2.substring(0,5);  
        System.out.println("The 2nd substring is: " + b);  
  
        // concat() method concatenates one string  
        // to the end of another string.  
        a=a.concat(b);  
        System.out.println("The resultant string is: " + " " + a);  
    }  
}
```

}

Output:

Enter the 1st string

ITER is my College.

The 1st string: ITER is my College.

Enter the 2nd string

ITER is in SOA.

The 2nd string: ITER is in SOA

The 1st substring is: ITER is

The 2nd substring is: ITER

The resultant string is: ITER is ITER

Breaking Strings Into Words

- Take a string and divide the string into words or tokens.
- Two methods:
 - **Split method:** split()
 - The string split() method breaks a given string around matches of the given regular expression.
 - Regular expressions are " ", ",", "-", ";", ":"
 - **String Tokenizer method:** StringTokenizer()
 - **The java.util.StringTokenizer class** allows you to break a string into tokens.
 - It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc.
 - construct a StringTokenizer around your string and call its methods hasMoreTokens() and nextToken().
 - hasMoreElements() and nextElement().

- There are 3 constructors defined in the StringTokenizer class.
 - *StringTokenizer(String str)*
 - Creates StringTokenizer with specified string.
 - *StringTokenizer(String str, String delim)*
 - Creates StringTokenizer with specified string and delimiter.
 - *StringTokenizer(String str, String delim, boolean returnValue)*
 - Creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Example: split() method

```
import java.util.*;
import java.util.Scanner;

public class StringDemo {

    public static void main(String[] args) {

        Scanner S = new Scanner(System.in);
        System.out.println("Enter the 1st string ");
        String s1 = S.nextLine();

        System.out.println("The 1st string: " + s1);

        for (String word : s1.split(" ")) {      //", ", "-", ",", "."
            System.out.println(word);
        }
    }
}
```

Output:

Enter the 1st string
ITER is My College.
The 1st string: ITER is My College.
ITER
is
My
College.

Example: StringTokenizer(String str) method

```
import java.util.*;  
import java.util.Scanner;  
  
public class StringDemo {  
  
    public static void main(String[] args) {  
  
        Scanner S = new Scanner(System.in);  
        System.out.println("Enter the 1st string ");  
        String s1 = S.nextLine();  
  
        System.out.println("The 1st string: " + s1);  
  
        //StringTokenizer st = new StringTokenizer(s1);  
  
        StringTokenizer st = new StringTokenizer(s1, ",");  
  
        while (st.hasMoreTokens())  
            System.out.println("Token: " + st.nextToken());  
    }  
}
```

Output:

Enter the 1st string

ITER, is, my, college.

The 1st string: ITER is my college.

Token: ITER

Token: is

Token: my

Token: college.

Example: *StringTokenizer(String str, String delim)*

```
import java.util.*;
public class StringDemo {

    public static void main(String[] args) {

        // TODO Auto-generated method stub

StringTokenizer st = new StringTokenizer("Hello, World|of|Java", ", |");
        while (st.hasMoreElements( ))
            System.out.println("Token: " + st.nextElement( ));
    }

}
```

Output:

Hello, World|of|Java

Token: Hello

Token: World

Token: of

Token: Java

Example: *StringTokenizer(String str, String delim, boolean returnValue)*

```
import java.util.*;
public class StringDemo {

    public static void main(String[] args) {

        // TODO Auto-generated method stub
```

```

StringTokenizer st = new StringTokenizer("Hello,
                                         World|ofJava", ", |", true);
while (st.hasMoreElements( ))
    System.out.println("Token: " + st.nextElement( ));
}

}

```

Output:

```

Token: Hello
Token: ,
Token:
Token: World
Token: |
Token: of
Token: |
Token: Java

```

Example: Write a program which reads a string and process it, if consecutive delimiter comes ignore it else print the token.

```

import java.util.*;

public class StrTokDemo4 {
    public final static int MAXFIELDS = 5;
    public final static String DELIM = "|";

    /** Processes one String, returns it as an array of Strings */
    public static String[] process(String line) {
        String[] results = new String[MAXFIELDS]; //array declaration

        // Unless you ask StringTokenizer to give you the tokens,
        // it silently discards multiple null tokens.
        StringTokenizer st = new StringTokenizer(line, DELIM, true);
        int i = 0;
        // stuff each token into the current slot in the array.
        while (st.hasMoreTokens()) {
            String s = st.nextToken();
            if (s.equals(DELIM)) {

```

```

if (i++>=MAXFIELDS)
    // This is messy: See StrTokDemo4b which uses
    // a List to allow any number of fields.
    throw new IllegalArgumentException("Input line " +
        line + " has too many fields");
    continue;
}
results[i] = s;
}
return results;
}

public static void printResults(String input, String[] outputs) {
    System.out.println("Input: " + input);
    int i=0;
    for (String s : outputs) {
        System.out.println("Output " + i + " was: " + s);
        ++i;
    }
}

public static void main(String[] args) {
    printResults("ABCD|BC|BC|D ", process("ABCD|BC|BC|D"));
    printResults("AB||C|D", process("AB||C|D"));
    printResults("AB||D|E", process("AB||D|E"));
}
}

```

Output:

Input: ABCD|B|C|D

Output 0 was: ABCD

Output 1 was: BC

Output 2 was: BC

Output 3 was: D

Output 4 was: null

Input: AB||C|D

Output 0 was: AB

Output 1 was: null

Output 2 was: null

Output 3 was: C

Output 4 was: D

Input: AB||D|E

Output 0 was: AB

Output 1 was: null

Output 2 was: null

Output 3 was: D

Output 4 was: E

Example: Write a program using java to read a string and find the number of words present in that string.

```
import java.util.*;
import java.util.Scanner;

public class StringDemo {

    public static void main(String[] args) {

        Scanner S = new Scanner(System.in);
        System.out.println("Enter the 1st string ");
        String s1 = S.nextLine();

        System.out.println("The 1st string: " + s1);

        StringTokenizer st = new StringTokenizer(s1);
        //StringTokenizer st = new StringTokenizer(s1, " ");

        int i=0;
        while (st.hasMoreTokens() ) {
            System.out.println("Token: " + st.nextToken());
            ++i;
        }

        System.out.println("Total number of words are " + i);

    }
}
```

Output:

Enter the 1st string

ITER is my college.

The 1st string: ITER is my college.

Token: ITER

Token: is

Token: my

Token: college.

Total number of words are 4

Example: Write a program to read a string whose words are separated by ',' find the number of words and number of ',' present in that string. The input string is: **Hello, World,of,Java.**

```
import java.util.*;
import java.util.Scanner;

public class StringDemo {

    public static void main(String[] args) {

        final String DELIM = ",";
        Scanner S = new Scanner(System.in);
        System.out.println("Enter the 1st string ");
        String s1 = S.nextLine();

        System.out.println("The 1st string: " + s1);

        int i=0,j=0;
        StringTokenizer st = new StringTokenizer(s1, ",", true);
        while (st.hasMoreElements( )) {
            Object s = st.nextElement();
            System.out.println("Token: " + s);
            if(s.equals(DELIM)) {
                ++i;
            }
            else
                ++j;
        }
    }
}
```

```
        }  
  
        System.out.println("Total number of Delimeter are " + i);  
        System.out.println("Total number of Words are " + j);  
    }  
}
```

Output:

Enter the 1st string
Hello, World,of,Java.

The 1st string: Hello, World,of,Java.

Token: Hello

Token: ,

Token: World

Token: ,

Token: of

Token: ,

Token: Java.

Total number of Delimeter are 3

Total number of Words are 4

Java StringBuffer class

- The Java StringBuffer class is used to create mutable (modifiable) string.
- The StringBuffer class in java is same as the String class except it is mutable i.e. it can be changed.

- The StringBuffer class is Synchronized .
 - Java StringBuffer class is thread-safe, i.e., multiple threads cannot access it simultaneously. So it is safe and will result in an order.
- Important methods of StringBuffer class are
 - 1) StringBuffer append() method
 - 2) StringBuffer insert() method
 - 3) StringBuffer replace() method
 - 4) StringBuffer delete() method
 - 5) StringBuffer reverse() method

1) StringBuffer append() method

The append() method concatenates the given argument with this string.

Example:

```
class StringBufferExample{

    public static void main(String args[]){

        StringBuffer sb=new StringBuffer("Hello ");
        System.out.println(sb); //prints Hello

        sb.append("Java ");           //now original string is changed
        sb.append("is language");

        System.out.println(sb); //prints Hello Java is language
    }
}
```

Output:

Hello Java is language

2) StringBuffer insert() method

The insert() method inserts the given string with this string at the given position.

Example:

```
class StringBufferExample2{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello ");  
        sb.insert(1,"Java");      //now original string is changed  
        System.out.println(sb);  //prints HJavaello  
    }  
}
```

Output:

HJavaello

3) StringBuffer replace() method

The replace() method replaces the given string from the specified beginIndex and endIndex. The syntax is :

replace(int beginIndex, int endIndex, string String)

Example:

```
class StringBufferExample3{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello");  
        sb.replace(1,3,"Java");  
    }  
}
```

```
        System.out.println(sb);           //prints HJavaLo  
    }  
}
```

Output:

HJavaLo

4) StringBuffer delete() method

The delete() method of StringBuffer class deletes the string from the specified beginIndex to endIndex.

delete(int beginIndex, int endIndex)

Example:

```
class StringBufferExample4{  
    public static void main(String args[]){  
        StringBuffer sb=new StringBuffer("Hello");  
        sb.delete(1,3);  
        System.out.println(sb);    //prints Hlo  
    }  
}
```

Output:

Hlo

5) StringBuffer reverse() method

The reverse() method of StringBuilder class reverses the current string.

```
class StringBufferExample5{
```

```
public static void main(String args[]){
    StringBuffer sb=new StringBuffer("Hello");
    sb.reverse();
    System.out.println(sb); //prints olleH
}
}
```

Output:

olleH

Java StringBuilder class

- The Java StringBuilder class is used to create mutable (modifiable) string.
- The Java StringBuilder class is same as StringBuffer class except that it is non-synchronized.
- It has been available since JDK 1.5.
- Important methods of StringBuilder class are
 - 1) StringBuilder append() method
 - 2) StringBuilder insert() method
 - 3) StringBuilder replace() method
 - 4) StringBuilder delete() method
 - 5) StringBuilder reverse() method

1) StringBuilder append() method

The StringBuilder append() method concatenates the given argument with this string.

Example:

```
class StringBuilderExample{  
  
    public static void main(String args[]){  
  
        StringBuilder sb=new StringBuilder("Hello ");  
  
        sb.append("Java");//now original string is changed  
  
        System.out.println(sb);//prints Hello Java  
  
    }  
  
}
```

Output:

Hello Java

Example:

```
public class StringBuilderDemo {  
  
    public static void main(String[] args) {  
  
        //using + operator  
        String s1 = "Hello" + ", " + "World";  
        System.out.println(s1);  
  
        // Build a StringBuilder, and append some things to it.  
        StringBuilder sb2 = new StringBuilder();  
        sb2.append("Hello");  
        sb2.append(',');  
        sb2.append(' ');  
        sb2.append("World");  
  
        // Get the StringBuilder's value as a String, and print it.  
        String s2 = sb2.toString();  
        System.out.println(s2);  
    }  
}
```

```

// Now do the above all over again, but in a more
// concise (and typical "real-world" Java) fashion.
System.out.println(
    new StringBuilder()
        .append("Hello")
        .append(',')
        .append(' ')
        .append("World"));
}

}

```

Output:

Hello, World
Hello, World
Hello, World

2) StringBuilder insert() method

The StringBuilder insert() method inserts the given string with this string at the given position.

Example:

```

class StringBuilderExample2{

    public static void main(String args[]){

        StringBuilder sb=new StringBuilder("Hello ");

        sb.insert(1,"Java");           //now original string is changed

        System.out.println(sb);       //prints HJavaello

    }

}

```

Output:

HJavaello

3) StringBuilder replace() method

The StringBuilder replace() method replaces the given string from the specified beginIndex and endIndex.

Example:

```
class StringBuilderExample3{  
  
    public static void main(String args[]){  
  
        StringBuilder sb=new StringBuilder("Hello");  
  
        sb.replace(1,3,"Java");  
  
        System.out.println(sb); //prints HJavaLo  
  
    }  
  
}
```

Output:

HJavaLo

4) StringBuilder delete() method

The delete() method of StringBuilder class deletes the string from the specified beginIndex to endIndex.

Example:

```
class StringBuilderExample4{  
  
    public static void main(String args[]){  
  
        StringBuilder sb=new StringBuilder("Hello");  
  
        sb.delete(1,3);  
  
        System.out.println(sb); //prints Hlo
```

```
    }  
}  
}
```

Output:

Hlo

5) StringBuilder reverse() method

The reverse() method of StringBuilder class reverses the current string.

Example:

```
class StringBuilderExample5{  
  
    public static void main(String args[]){  
  
        StringBuilder sb=new StringBuilder("Hello");  
  
        sb.reverse();  
  
        System.out.println(sb);//prints olleH  
  
    }  
  
}
```

Output:

olleH

Example: Write a program using a StringBuilder, consider the need to convert a list of items into a comma-separated list, while avoiding getting an extra comma after the last element of the list. (NOTE: Using split() method).

```
public class StringBuilderDemo {  
  
    public static void main(String[] args) {  
  
        String SAMPLE_STRING = "ITER is My College.";
```

```

System.out.println("The Input String is " + SAMPLE_STRING);

StringBuilder sb1 = new StringBuilder();

for (String word : SAMPLE_STRING.split(" ")) {

    if (sb1.length() > 0) {
        sb1.append(", ");
    }
    sb1.append(word);
}
System.out.println("The Input String is " + sb1);
}
}

```

Output:

The Input String is ITER is My College.
The Output String is ITER, is, My, College.

Example: Write a program using a StringBuilder, consider the need to convert a list of items into a comma-separated list, while avoiding getting an extra comma after the last element of the list. (NOTE: Using StringTokenizer() method).

```

import java.util.*;
public class StringBuilderDemo {

    public static void main(String[] args) {

        String SAMPLE_STRING = "ITER is My College.";
        System.out.println("The Input String is " + SAMPLE_STRING);

        // Method using a StringTokenizer
        StringTokenizer st = new StringTokenizer(SAMPLE_STRING);

        StringBuilder sb3 = new StringBuilder();
        while (st.hasMoreElements()) {

```

```

sb3.append(st.nextToken());

if (st.hasMoreElements()) {

    sb3.append(", ");
}

System.out.println(sb3);
}
}

```

Output:

The Input String is ITER is My College.
The Output String is ITER, is, My, College.

Reversing a String by Character

Example: reverse a string by character easily, using charAt() method.

```

import java.util.*;
public class ReverseString {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String original, reverse = "";
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter a string to reverse");
        original = sc.nextLine();

        int length = original.length();

        for (int i = length - 1 ; i >= 0 ; i--)
            reverse = reverse + original.charAt(i);

        System.out.println("Reverse of the string: " + reverse);
    }
}

```

```
    }  
}
```

Output:

Enter a string to reverse

ITER is My College.

Reverse of the string: .egelloC yM si RETI

Example: To reverse the characters in a string, using the StringBuilder reverse() method.

```
import java.util.*;  
public class ReverseString {  
  
    public static void main(String[] args) {  
  
        String original, reverse = "";  
  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Enter a string to reverse");  
        original = sc.nextLine();  
  
        StringBuilder sb=new StringBuilder(original);  
        System.out.println("Reverse of the string: " + sb.reverse());  
    }  
}
```

Output:

Enter a string to reverse

ITER is My College.

Reverse of the string: .egelloC yM si RETI

Reversing a String by Word

- To reverse the string by word is a 2 step process:

- From the beginning of the string each word has been taken and adds into a Stack.
- Then retrieve the words from stack in LIFO order.

Example:

```

import java.util.*;

public class ReverseString {

    public static void main(String[] args) {

        String original, revword="";

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a string to reverse");
        original = sc.nextLine();

        // Put it in the stack frontwards
        Stack<String> myStack = new Stack<>();           //Create the stack

        StringTokenizer st = new StringTokenizer(original);

        while (st.hasMoreTokens()) {
            String word = st.nextToken();
            myStack.push(word);                         //Insert into the stack
        }

        // Print the stack backwards
        System.out.print(" " + original + " " + " backwards by word is:\n\t");

        while (!myStack.empty()) {
            String revword = myStack.pop();           //Retrieve from the stack
            System.out.print(revword);
            System.out.print(" ");
        }
        System.out.println("");
    }
}

```

Output:

"ITER is My College." backwards by word is:
"College. My is ITER "

Java String equals() Method

- The **java string equals()** method compares the two given strings based on the content of the string.
- If any character is not matched, it returns false. If all characters are matched, it returns true.

Example:

```
public class StringEquals {  
  
    public static void main(String[] args) {  
  
        String s1="java";  
        String s2="java";  
        String s3="JAVA";  
        String s4="python";  
  
        System.out.println(s1==s2);      //true because content and case is same  
  
        System.out.println(s1.equals(s2)); //true because content and case is  
                                         same  
        System.out.println(s1.equals(s3)); //false because case is not same  
  
        System.out.println(s1.equals(s4)); //false because content is not same  
  
    }  
}
```

Output:

true
true
false

false

Java String equalsIgnoreCase() Method

- The **String equalsIgnoreCase()** method compares the two given strings on the basis of content of the string irrespective of case of the string.
- It is like equals() method but doesn't check case.
- If any character is not matched, it returns false otherwise it returns true.

Example:

```
public class StringEquals {  
  
    public static void main(String[] args) {  
  
        String s1="java";  
        String s2="java";  
        String s3="JAVA";  
        String s4="python";  
  
        System.out.println(s1==s2);  
  
        System.out.println(s1.equalsIgnoreCase(s2));  
  
        System.out.println(s1.equalsIgnoreCase(s3));  
  
        System.out.println(s1.equalsIgnoreCase(s4));  
  
    }  
  
}
```

Output:

true
true
true

false

Java String trim()

- The **java string trim()** method eliminates leading and trailing spaces.
- The unicode value of space character is '\u0020'.
- The trim() method in java string checks this unicode value before and after the string, if it exists, then removes the spaces and returns the omitted string.

Example:

```
public class StringTrim {  
  
    public static void main(String[] args) {  
  
        String s1 =" hello java string ";  
        System.out.println("Length of String before trim: " + s1.length());  
        System.out.println(s1);           //Without trim()  
  
        String tr = s1.trim();  
        System.out.println("Length of String after trim: " +tr.length());  
        System.out.println(tr);         //With trim()  
    }  
}
```

Output:

Length of String before trim: 22

hello java string

Length of String after trim: 17

hello java string

Entering Nonprintable Characters

- Put nonprintable characters into strings.

To get:	Use:	Notes
Tab	\t	
Linefeed (Unix newline)	\n	The call System.getProperty("line.separator") will give you the platform's line end.
Carriage return	\r	
Form feed	\f	
Backspace	\b	
Single quote	\'	
Double quote	\"	
Unicode character	\uNNNN	Four hexadecimal digits (no \x as in C/C++). See http://www.unicode.org for codes.
Octal(!) character	\NNN	Who uses octal (base 8) these days?
Backslash	\\"	

Example:

```
public class StringEscapes {
    public static void main(String[] argv) {
        System.out.println("Java Strings in action:");
        System.out.println("A tab key: \t(what comes after)");
        System.out.println("A newline: \n(what comes after)");
        System.out.println("A UniCode character: \u0207");
        System.out.println("A backslash character: \\");
    }
}
```

Output:

Java Strings in action:
A tab key: (what comes after)
A newline:
(what comes after)
A UniCode character: ?

A backslash character: \

Expanding and Compressing Tabs

- Convert space characters to tab characters in a string, or vice versa.

Example: Write a program to replace all space in a string by tabs. Note: Here we have used **replaceall()** method. The **replaceall()** requires two parameters, both are string types.

```
import java.util.*;
public class StringReplace {

    public static void main(String[] args) {

        String str2 = "ITER is My College";
        System.out.println("Input string: " + str2);

        str2 = str2.replaceAll("\\s", "\t");
        System.out.println("Output string: " + str2);
    }
}
```

Output:

Input string: ITER is My College
ITER is My College

Example: Write a program to replace all tab in a string by spaces.

```
import java.util.*;
public class StringReplace {

    public static void main(String[] args) {

        String str2 = "ITER      is      My      College";
        System.out.println("Input string: " + str2);
    }
}
```

```
        str2 = str2.replaceAll("\t", " ");
        System.out.println("Output string: "+ str2);

    }
}
```

Output:

Input string: ITER is My College
Output string: ITER is My College

Example: Write a program to //replaces all occurrences of "a" to "e".

```
public class ReplaceAllExample1{

    public static void main(String args[]){

        String s1="java is a very good language";

        String replaceString=s1.replaceAll("a","e");

        System.out.println(replaceString);

    }
}
```

Output:

jeve is e very good lenguege

Example: Write a program to replaces particular word by using replace all.

```
public class ReplaceAllExample2{

    public static void main(String args[]){

        String s1="C is a very good language";

        String replaceString=s1.replaceAll("C","java");

    }
}
```

```
        System.out.println(replaceString);  
    }  
}
```

Output:

java is a very good language

Soundex Name Comparisons

- The Soundex algorithm was developed by Robert Russell in 1910 for the words in English.
- The main principle behind this algorithm is that consonants are grouped depending on the ordinal numbers and finally encoded into a value against which others are matched.
- It aims to find a code for every word by above process which is called soundex code.
- The Soundex code is 4-character code:
 - the letter is the first letter of the name, and
 - the digits encode the remaining consonants.

Algorithm to find soundex code is as below:

1. Retain the first letter of the name.
2. Change all occurrence of following letter to zero.

a, e, i, o, u, y, h, w → 0

3. Replace consonants with digits as follows:

b, f, p, v → 1
c, g, j, k, q, s, x, z → 2
d, t → 3
l → 4

$m, n \rightarrow 5$
 $r \rightarrow 6$

4. Remove all pairs of consecutive digits.
5. Remove all zeros from the resulting string.
6. If the resultant code size is more than 4, then just retain the first 4 by truncating rest code.
7. If the resultant code size is less than 4, then append with zeros.

Example: Soundex Algorithm example.

```
public class Soundex1 {  
  
    public static String getCode(String s) {  
  
        char[] x = s.toUpperCase().toCharArray();  
  
        char firstLetter = x[0];  
  
        //RULE [ 2 ]  
        //Convert letters to numeric code  
        for (int i = 0; i < x.length; i++)  
        {  
            switch (x[i])  
            {  
                case 'B':  
                case 'F':  
                case 'P':  
                case 'V': {  
                    x[i] = '1';  
                    break;  
                }  
  
                case 'C':  
                case 'G':  
                case 'J':  
                case 'K':  
                case 'Q':  
                case 'S':  
                case 'X':
```

```

case 'Z': {
    x[i] = '2';
    break;
}

case 'D':
case 'T': {
    x[i] = '3';
    break;
}

case 'L': {
    x[i] = '4';
    break;
}

case 'M':
case 'N': {
    x[i] = '5';
    break;
}

case 'R': {
    x[i] = '6';
    break;
}

default: {
    x[i] = '0';
    break;
}
}

//Return first letter of word
String output = " " + firstLetter;

// consonants coding by removing duplicate digits and zero
for (int i = 1; i < x.length; i++)

```

```

if (x[i] != x[i - 1] && x[i] != '0')
    output += x[i];

//Pad with 0's or truncate

output = output + "0000";

return output.substring(0, 4);
}

public static void main(String[] args) {
    // TODO Auto-generated method stub
    String name1 = "beer";
    String name2 = "bear";
    String name3 = "bearer";

    System.out.println(Soundex1.getCode(name1) + ' ' + name1);
    System.out.println(Soundex1.getCode(name2) + ' ' + name2);
    System.out.println(Soundex1.getCode(name3) + ' ' + name3);
}
}

```

Output:

B600 beer
B600 bear
B660 bearer

Example: Soundex Algorithm example.

```

public class Soundex {
    //static boolean debug = false;
    /* Implements the mapping
     * from: AEHIOWYWBFPCVCGJKQSXZDTLMNR
     * to: 000000011112222222334556
     */
}

```

```

public static final char[] MAP = {
//A B C D E F G H I J K L M
'0', '1', '2', '3', '0', '1', '2', '0', '0', '2', '2', '4', '5',
//N O P Q R S T U V W X Y Z
'5', '0', '1', '2', '6', '2', '3', '0', '1', '0', '2', '0', '2'};

public static String soundex(String s) {
    // Algorithm works on uppercase (mainframe era).
    String t = s.toUpperCase();
    StringBuffer res = new StringBuffer();
    char c, prev = '?', prevOutput = '?';
    // Main loop: find up to 4 chars that map.

    for (int i=0; i<t.length() && res.length() < 4 &&
        (c = t.charAt(i)) != ';' i++) {
        // Check to see if the given character is alphabetic.
        // Text is already converted to uppercase. Algorithm
        // only handles ASCII letters, do NOT use Character.isLetter()!
        // Also, skip double letters.
        if (c>='A' && c<='Z' && c != prev) {
            prev = c;
            // First char is installed unchanged, for sorting.
            if (i==0) {
                res.append(c);
            }
            else
            {
                char m = MAP[c-'A'];
                //if (debug) {
                //    System.out.println(c + " --> " + m);
                //}
                if (m != '0' && m != prevOutput) {
                    res.append(m);
                    prevOutput = m;
                }
            }
        }
    }
    if (res.length() == 0)
        return null;
}

```

```

for (int i=res.length(); i<4; i++)
    res.append('0');
return res.toString();
}

public static void main(String[] args) {
    // TODO Auto-generated method stub

    String[] names = {
        "Darwin, Ian",
        "Davidson, Greg",
        "Darwent, William",
        "Derwin, Daemon",
        "Der"
    };

    for (String name : names) {
        System.out.println(Soundex.soundex(name) + ' ' + name);
    }
}

```

Output:

D650 Darwin, Ian
D132 Davidson, Greg
D653 Darwent, William
D650 Derwin, Daemon
D600 Der

4. Write a program to enter a string and count the frequency of each character present in it.

```

public class Frequency

{
    public static void main(String[] args) {

```

```
String str = "picture perfect";  
  
int[] freq = new int[str.length()];  
  
int i, j;  
  
//Converts given string into character array  
  
char string[] = str.toCharArray();  
  
for(i = 0; i <str.length(); i++) {  
  
    freq[i] = 1;  
  
    for(j = i+1; j <str.length(); j++) {  
  
        if(string[i] == string[j]) {  
  
            freq[i]++;  
  
            //Set string[j] to 0 to avoid printing visited character  
  
            string[j] = '0';  
  
        }  
  
    }  
  
}  
  
//Displays the each character and their corresponding frequency  
  
System.out.println("Characters and their corresponding frequencies");  
  
for(i = 0; i <freq.length; i++) {  
  
    if(string[i] != ' ' && string[i] != '0')  
  
        System.out.println(string[i] + "-" + freq[i]);
```

```
    }  
}  
}
```

Output:

Characters and their corresponding frequencies

p-2
i-1
c-2
t-2
u-1
r-2
e-3
f-1

6. Write a program to enter N number of strings, arrange them in ascending order.

```
java.util.Scanner;  
public class Ch3Ex6 {  
  
public static void main(String[] args) {  
    int count;  
    String temp;  
    Scanner scan = new Scanner(System.in);  
  
    System.out.print("Enter number of strings: ");  
    count = scan.nextInt();  
  
    System.out.println("Enter the " + count + " Strings one by one:");  
    String str[] = new String[count];  
    Scanner scan2 = new Scanner(System.in);  
  
for(int i = 0; i < count; i++)  
{  
    str[i] = scan2.nextLine();
```

```

        }
        scan.close();
        scan2.close();

//Sorting the strings
for (int i = 0; i < count; i++) {
{
    for (int j = i + 1; j < count; j++) {
        if (str[i].compareTo(str[j])>0)
        {
            temp = str[i];
            str[i] = str[j];
            str[j] = temp;
        }
    }
}
}

//Displaying the strings after sorting
System.out.println("Strings in Sorted Order:");
for (int i = 0; i <= count - 1; i++)
{
    System.out.println(str[i]);
}

}

```

Output:

Enter number of strings: 5

Enter the 5 Strings one by one:

cbi
bca
mca
xyz
aaa

Strings in Sorted Order:

aaa
bca
cbi

mca
xyz

10. The global replace function is a string-processing algorithm found in every word processor. Write a method that takes three String arguments: a document, a target string, and a replacement string. The method should replace every occurrence of the target string in the document with the replacement string. For example, if the document is “To be or not to be, that is the question,” and the target string is “be,”, and the replacement string is “see,” the result should be, “To see or not to see, that is the question.”

Solutions:

```
import java.util.*;
public class StringProblem {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner S = new Scanner(System.in);
        System.out.println("Enter the original string ");
        String s1 = S.nextLine();

        //StringBuilder s2 = new StringBuilder(s1);
        System.out.println("Enter the target string: ");
        String target = S.nextLine();
        System.out.println("Enter the replacement string ");
        String replacement = S.nextLine();

        System.out.println(" ");
        System.out.println("The original string is: " + s1);
        System.out.println("The target string is: " + target);
        System.out.println("The replacement string is: " +
                           replacement);

        String result = s1.replaceAll(target, replacement);
        System.out.println("The resultant string is: " + result);

    }
}
```

Output:

Enter the original string

To be or not to be, that is the questions.

Enter the target string:

be

Enter the replacement string

see

The original string is: To be or not to be, that is the questions.

The target string is: be

The replacement string is: see

The resultant string is: To see or not to see, that is the questions.

12. Write a method that converts its String parameter so that letters are written in blocks five characters long. For example, consider the following two versions of the same sentence:

This is how we would ordinarily write a sentence.

Thisi showww ewoul dordi naril ywrit easen tence

Solution:

```
import java.util.Scanner;
import java.util.*;
public class NumberToWord {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        Scanner S = new Scanner(System.in);
        System.out.println("Enter the original string ");
        String s1 = S.nextLine();
        System.out.println("The original string is: "+ s1);

        String s2 = s1.replaceAll("\\s", "");
        System.out.println("After replacing space, string is: "+ s2);
        int count = 0;
```

```

//StringBuilder s3 = new StringBuilder();
//s3.append(s2);

System.out.print("the resultant string is: ");
char ch;
for(int i=0;i<=s2.length()-1;i++)
{
    count++;
    if(count == 6)
    {
        ch = ' ';
        System.out.print(ch);
        count=1;
        ch = s2.charAt(i);
        System.out.print(ch);
    }
    else
    {
        ch = s2.charAt(i);
        System.out.print(ch);
    }
}
}

```

Output:

The original string is: This is how we would ordinarily write a sentence.

After replacing space, string is: Thisishowwewouldordinarilywriteasentence.

The resultant string is: Thisi showw ewoul dordi naril ywrit easen tence .

25. Write an program to print all permutations of a given String in Java. For example, if given input is "123" then your program should print all 6 permutations e.g. "123", "132", "213", "231", "312" and "321".

Solution:

```
import java.util.Scanner;
```

```

public class Ch3Ex25 {
    //Function for swapping the characters at position I with character at position
    j
    public static String swapString(String a, int i, int j) {
        char[] b = a.toCharArray();
        char ch;
        ch = b[i];
        b[i] = b[j];
        b[j] = ch;
        return String.valueOf(b);
    }

    public static void main(String[] args)
    {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter the strings: ");
        String str = scan.nextLine();

        int len = str.length();
        System.out.println("All the permutations of the string are: ");
        generatePermutation(str, 0, len);
    }
}

//Function for generating different permutations of the string
public static void generatePermutation(String str, int start, int end)
{
    //Prints the permutations
    if (start == end-1)
        System.out.println(str);
    else
    {
        for (int i = start; i < end; i++)
        {
            //Swapping the string by fixing a character
            str = swapString(str,start,i);
            //Recursively calling function generatePermutation() for rest of the
            characters
            generatePermutation(str,start+1,end);
            //Backtracking and swapping the characters again.
            str = swapString(str,start,i);
        }
    }
}

```

```
        }
    }
}
}
```

Output:

Enter the strings: 123

All the permutations of the string are:

```
123
132
213
231
321
312
```

26. Write a Java program which will take a String input and print out a number of vowels and consonants on that String. For example, if the input is “Java” then your program should print “2 vowels and 2 consonants”.

Solution:

```
import java.util.Scanner;
public class Ch3Ex26 {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Enter the strings: ");
        String str = scan.nextLine();

        int vowels = 0, consonants = 0, digits = 0, spaces = 0;
        str = str.toLowerCase();

        for(int i = 0; i < str.length(); ++i)
        {
            char ch = str.charAt(i);
            if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u')
            {
                ++vowels;
            }
            else if((ch >= 'a' && ch <= 'z'))
            {
                ++consonants;
            }
        }
    }
}
```

```
        }
    else if( ch >= '0' && ch <= '9')
    {
        ++digits;
    }
else if (ch == ' ')
{
    ++spaces;
}
System.out.println("Vowels: " + vowels);
System.out.println("Consonants: " + consonants);
System.out.println("Digits: " + digits);
System.out.println("White spaces: " + spaces);
    }
}
```

Output:

Enter the strings: **java**

Vowels: 2

Consonants: 2

Digits: 0

White spaces: 0

Pattern Matching with Regular Expressions

What are regular expressions?

- A regular expression is a sequence of characters.
- The abbreviation for regular expression is *regex*.
- Java Regular Expressions Package, `java.util.regex.*`
- A *regular expression* defines a search pattern for strings.
- The search pattern can be anything, such as,
 - a simple character, or
 - a fixed string or
 - a complex expression containing special characters describing the pattern.
- The pattern defined by the regex may match one or several times or not at all for a given string.
- Regular expressions can be used to search, edit and manipulate string (“find and replace”-like operations.), URL matching, etc.
- The pattern defined by the regex is applied on the text from left to right.
 - For example, the regex aba will match **ababababa** only two times (aba_aba__).

Regular expression metacharacter syntax

<u>Subexpression</u>	<u>Matches</u>	<u>Notes</u>
----------------------	----------------	--------------

General

\^	Start of line/string
\\$	End of line/string
\b	Word boundary
\B	Not a word boundary
\A	Beginning of entire string
\z	End of entire string
\Z	End of entire string (except allowable final line terminator)
.	Any one character (except line terminator)
[...]	“Character class”; any one character from those listed
[\^...]	Any one character not from those listed

Alternation and Grouping

(...)	Grouping (capture groups)
	Alternation
(?:_re_)	Noncapturing parenthesis
\G	End of the previous match

<u>Subexpression</u>	<u>Matches</u>	<u>Notes</u>
----------------------	----------------	--------------

\ n Back-reference to capture group number " n "

Normal (greedy) quantifiers

{ m,n } Quantifier for “from m to n repetitions”

{ $m ,$ } Quantifier for " m or more repetitions"

{ m } Quantifier for “exactly m repetitions”

{ $,n$ } Quantifier for 0 up to n repetitions

* Quantifier for 0 or more repetitions Short for {0,}

+ Quantifier for 1 or more repetitions Short for {1,}

? Quantifier for 0 or 1 repetitions
 (i.e., present exactly once, or not at all) Short for {0,1}

Reluctant (non-greedy) quantifiers

{ m,n }? Reluctant quantifier for “from m to n repetitions”

{ $m ,$ }? Reluctant quantifier for " m or more repetitions"

{ $,n$ }? Reluctant quantifier for 0 up to n repetitions

*? Reluctant quantifier: 0 or more

+? Reluctant quantifier: 1 or more

?? Reluctant quantifier: 0 or 1 times

<u>Subexpression</u>	<u>Matches</u>	<u>Notes</u>
----------------------	----------------	--------------

Possessive (very greedy) quantifiers

{ <i>m,n</i> }+	Possessive quantifier for “from <i>m</i> to <i>n</i> repetitions”
{ <i>m</i> ,}+	Possessive quantifier for “ <i>m</i> or more repetitions”
{, <i>n</i> }+	Possessive quantifier for 0 up to <i>n</i> repetitions
*+	Possessive quantifier: 0 or more
++	Possessive quantifier: 1 or more
?+	Possessive quantifier: 0 or 1 times

Escapes and shorthands

\	Escape (quote) character: turns most metacharacters off; turns subsequent alphabetic into metacharacters
\Q	Escape (quote) all characters up to \E
\E	Ends quoting begun with \Q
\t	Tab character
\r	Return (carriage return) character
\n	Newline character
\f	Form feed

<u>Subexpression</u>	<u>Matches</u>	<u>Notes</u>
\w	Character in a word	Use \w+ for a word;
\W	A nonword character	
\d	Numeric digit	Use \d+ for an integer;
\D	A nondigit character	
\s	Whitespace	Space, tab, etc., as determined by java.lang.Character.isWhitespace()
\S	A nonwhitespace character	

Unicode blocks (representative samples)

\p{InGreek}	A character in the Greek block	(Simple block)
\P{InGreek}	Any character not in the Greek block	
\p{Lu}	An uppercase letter (Simple category)	
\p{Sc}	A currency symbol	

POSIX-style character classes (defined only for US-ASCII)

\p{Alnum}	Alphanumeric characters	[A-Za-z0-9]
\p{Alpha}	Alphabetic characters	[A-Za-z]
\p{ASCII}	Any ASCII character	[\x00-\x7F]
\p{Blank}	Space and tab characters	
\p{Space}	Space characters	[\t\n\x0B\f\r]
\p{Cntrl}	Control characters	[\x00-\x1F\x7F]

<u>Subexpression</u>	<u>Matches</u>	<u>Notes</u>
\p{Digit}	Numeric digit characters	[0-9]
\p{Graph}	Printable and visible characters (not spaces or control characters)	
\p{Print}	Printable characters	
\p{Punct}	Punctuation characters	One of !"#\$%&'()* +,-./;:<=>?@[{}]\^_`{ }\~
\p{Lower}	Lowercase characters	[a-z]
\p{Upper}	Uppercase characters	[A-Z]
\p{XDigit}	Hexadecimal digit characters	[0-9a-fA-F]

How to write regular expression?

- **Repeaters : * , + and { } :**

- These symbols act as repeaters and tell the computer that the preceding character is to be used for more than just one time.
-
- **The asterisk symbol (*):**

It tells the computer to match the preceding character (or set of characters) for 0 or more times (upto infinite).

- **Example :** The regular expression ab*c will give ac, abc, abbc, abbbc....ans so on

-

- **The Plus symbol (+):**

It tells the computer to repeat the preceding character (or set of characters) for atleast one or more times(upto infinite).

- **Example :** The regular expression ab+c will give abc, abbc, abbbc, ... and so on.

- **The curly braces {...}:**

It tells the computer to repeat the preceding character (or set of characters) for as many times as the value inside this bracket.

- **Example :** {2} means that the preceding character is to be repeated 2 times,
- {min,} means the preceding character is matches min or more times.
- {min,max} means that the preceding character is repeated at least min & at most max times.

- **Wildcard – (.)**

The dot symbol can take place of any other symbol, that is why it is called the wildcard character.

- **Example :** The Regular expression .* will tell the computer that any character can be used any number of times.

- **Optional character – (?)**

This symbol tells the computer that the preceding character may or may not be present in the string to be matched.

- **Example :** We may write the format for document file as – “docx?”
 - The ‘?’ tells the computer that x may or may not be present in the name of file format.

- **The caret (^) symbol:**

Setting position for match : tells the computer that the match must start at the beginning of the string or line.

- **Example :** ^\d{3} will match with patterns like "901" in "901-333-".

- **The dollar (\$) symbol**
It tells the computer that the match must occur at the end of the string or before \n at the end of the line or string.
 - **Example :** -\d{3}\\$ will match with patterns like "-333" in "-901-333".
- **[set_of_characters]** – Matches any single character in set_of_characters. By default, the match is case-sensitive.
 - **Example :** [abc] will match characters a, b or c in any string.
- **[^set_of_characters] – Negation:** Matches any single character that is not in set_of_characters. By default, the match is case sensitive.
 - **Example :** [^abc] will match any character except a,b,c.
- **[first-last] – Character range:** Matches any single character in the range from first to last.
 - **Example :** [a-zA-z] will match any character from a to z or A to Z.
- **The Escape Symbol : **
If you want to match for the actual '+', '.' etc characters, add a backslash(\) before that character. This will tell the computer to treat the following character as a search character and consider it for matching pattern.
 - **Example :** \d+[\+-x]*\d+ will match patterns like "2+2" and "3*9" in "(2+2) * 3*9".
- **Grouping Characters ()**
A set of different symbols of a regular expression can be grouped together to act as a single unit and behave as a block, for this, you need to wrap the regular expression in the parenthesis().
 - **Example :** ([A-Z]\w+) contains two different elements of the regular expression combined together. This expression will match any pattern containing uppercase letter followed by any character.
- **Vertical Bar (|) :**
Matches any one element separated by the vertical bar (|) character.

- **Example :** th(e|is|at) will match words - the, this and that.

Example: There are three ways to write the regex example in Java.

```
import java.util.regex.*;

public class Regex {

    public static void main(String[] args) {

        //1st way
        Pattern p = Pattern.compile(".s"); // . represents single
                                         character
        Matcher m = p.matcher("as");
        boolean b = m.matches();

        //2nd way
        boolean b2=Pattern.compile(".s").matcher("as").matches();

        //3rd way
        boolean b3 = Pattern.matches(".s", "as");

        System.out.println(b+" "+b2+" "+b3);
    }
}
```

Output:

true true true

Example: Regular Expression . (dot), which represents a single character.

```
import java.util.regex.*;

public class Regex {

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

```

System.out.println(Pattern.matches(".s", "as"));
//true (2nd char is s)

System.out.println(Pattern.matches(".s", "mk"));
//false (2nd char is not s)

System.out.println(Pattern.matches(".s", "mst"));
//false (has more than 2 char)

System.out.println(Pattern.matches(".s", "amms"));
//false (has more than 2 char)

System.out.println(Pattern.matches(..s, "mas"));
//true (3rd char is s)
}

}

```

Output:

```

true
false
false
false
true

```

Example: Regular Expression Character classes

```

import java.util.regex.*;

public class Regex {

    public static void main(String[] args) {

        System.out.println(Pattern.matches("[amn]", "abcd"));
        //false (not a or m or n)

        System.out.println(Pattern.matches("[amn]", "a"));
        //true (among a or m or n)
    }
}

```

```

System.out.println(Pattern.matches("[amn]", "ammmna"));
//false (m and a comes more than once)

System.out.println(Pattern.matches("[^n].*", "abc")); //true

System.out.println(Pattern.matches("[^n]", "abc mnc"));
//false
}

}

```

Output:

false
true
false
true
false

Example: Regular Expression Character classes and Quantifiers

```

import java.util.regex.*;

public class Regex {

    public static void main(String[] args) {

        System.out.println("? quantifier ....");

        System.out.println(Pattern.matches("[amn]?", "a"));
//true (a or m or n comes one time)

        System.out.println(Pattern.matches("[amn]?", "aaa"));
//false (a comes more than one time)

        System.out.println(Pattern.matches("[amn]?", "aammmnn"));
//false (a m and n comes more than one time)

        System.out.println(Pattern.matches("[amn]?", "aazzta"));
//false (a comes more than one time)
    }
}

```

```

System.out.println(Pattern.matches("[amn]?", "am"));
//false (a or m or n must come one time)

System.out.println("+ quantifier ....");

System.out.println(Pattern.matches("[amn]+", "a"));
//true (a or m or n once or more times)

System.out.println(Pattern.matches("[amn]+", "aaa"));
//true (a comes more than one time)

System.out.println(Pattern.matches("[amn]+", "aammmnn"));
//true (a or m or n comes more than once)
System.out.println(Pattern.matches("[amn]+", "aazzta"));
//false (z and t are not matching pattern)

System.out.println("* quantifier ....");

System.out.println(Pattern.matches("[amn]*", "ammmnaa"));
//true (a or m or n may come zero or more times)
}
}

```

Output:

```

? quantifier ....
true
false
false
false
false
false
+ quantifier ....
true
true
true
false
* quantifier ....
true

```

Example: Regular Expression Metacharacters, \d and \D

```
import java.util.regex.*;  
  
public class Regex {  
  
    public static void main(String[] args) {  
  
        System.out.println("metacharacters d...."); //d means digit  
  
        System.out.println(Pattern.matches("\\d", "abc"));  
                                //false (non-digit)  
  
        System.out.println(Pattern.matches("\\d", "9"));  
                                //true (digit and comes once)  
  
        System.out.println(Pattern.matches("\\d", "4443"));  
                                //false (digit but comes more than once)  
  
        System.out.println(Pattern.matches("\\d", "323abc"));  
                                //false (digit and char)  
  
        System.out.println(Pattern.matches("\\d*", "3"));  
                                //true (digit and comes more than once)  
  
        System.out.println("metacharacters d with quantifier....");  
  
        System.out.println("metacharacters D....");  
                                //D means non-digit  
  
        System.out.println(Pattern.matches("\\D", "abc"));  
                                //false (non-digit but comes more than once)  
  
        System.out.println(Pattern.matches("\\D", "1"));  
                                //false (digit)  
  
        System.out.println(Pattern.matches("\\D", "4443"));  
                                //false (digit)
```

```

System.out.println(Pattern.matches("\D", "323abc"));
//false (digit and char)

System.out.println(Pattern.matches("\D", "m"));
//true (non-digit and comes once)

System.out.println("metacharacters D with quantifier....");

System.out.println(Pattern.matches("\D*", "mak"));
//true (non-digit and may come 0 or more times)
}

}

```

Output:

```

metacharacters d....
false
true
false
false
metacharacters d with quantifier....
true
metacharacters D....
false
false
false
false
true
metacharacters D with quantifier....
true

```

Example: Create a regular expression that accepts alphanumeric characters only. It's length must be six characters long only.

```

import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

```

```
System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun2"));
//true

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "kkvarun32"));
//false (more than 6 char)

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "JA2Uk2"));
//true

System.out.println(Pattern.matches("[a-zA-Z0-9]{6}", "arun$2"));
//false ($ is not matched)
}
```

Output:

true
false
true
false

Example: Create a regular expression that accepts 10 digit numeric characters starting with 7, 8 or 9 only.

```

System.out.println(Pattern.matches("[789][0-9]{9}",
                                "99530389490")); //false (11 characters)

System.out.println(Pattern.matches("[789][0-9]{9}",
                                "6953038949")); //false (starts from 6)

System.out.println(Pattern.matches("[789][0-9]{9}",
                                "8853038949")); //true

System.out.println("by metacharacters ...");

System.out.println(Pattern.matches("[789]{1}\d{9}",
                                "8853038949")); //true

System.out.println(Pattern.matches("[789]{1}\d{9}",
                                "3853038949")); //false (starts from 3)

}

}

```

Output:

```

by character classes and quantifiers ...
true
true
false
false
true
by metacharacters ...
true
false

```

Example: Subexpression: ^ Matches: Start of line/string

```

import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

```

```

String Regex = "^dog.*";
String Input = "dog xyz";

boolean match = Pattern.matches(Regex, Input);

System.out.print("Output: ");

if (match)
    System.out.print(" found");
else
    System.out.print(" not found");
}
}

```

Output: found

Example: Subexpression: \$ Matches: End of line/string

```

import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = ".*dog$";
        String Input = "abc dog";

        boolean match = Pattern.matches(Regex, Input);

        System.out.print("Output: ");

        if (match)
            System.out.print(" found");
        else
            System.out.print(" not found");
    }
}

```

Output: found

Example: Subexpression: \b Matches: Word boundary

Xyz abc pqr

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = ".*\\bdog\\b.*";
        String Input = "abc dog xyz";

        //String Input = "abcdoggxyz"; //Output: not found

        boolean match = Pattern.matches(Regex, Input);

        System.out.print("Output: ");
        if (match)
            System.out.print(" found");
        else
            System.out.print(" not found");

    }

}
```

Output: found

Example: Subexpression: \B Matches: Not a word boundary

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = ".*\\bdog\\B.*";
                    // \b word boundary \B not word boundary
        String Input = "abc dogpqr xyz";

    }

}
```

```

//String Regex = ".*\\Bgpq\\B.*";
//          // \B not word boundary
//String Input = "abcdogpqxyz";

boolean match = Pattern.matches(Regex, Input);

System.out.print("Output: ");
if (match)
    System.out.print(" found");
else
    System.out.print(" not found");
}
}

Output: found

```

Example: Subexpression: \A Matches: Beginning of entire string

```

import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = "\\Aabc.*";
        String Input = "abc dogpqr xyz";
        //String Regex = ".*\\Axxyz"; // Output: not found
        //String Input = "abcdogpqxyz";
        boolean match = Pattern.matches(Regex, Input);
        System.out.print("Output: ");
        if (match)
            System.out.print(" found");
        else
            System.out.print(" not found");
    }
}

Output: found

```

Example: Subexpression: \z Matches: End of entire string

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = ".*xyz\\z";
        String Input = "abc dogpqr xyz";

        boolean match = Pattern.matches(Regex, Input);

        System.out.print("Output: ");
        if (match)
            System.out.print(" found");
        else
            System.out.print(" not found");
    }
}
```

Output: found

Example: Subexpression: [...] Matches: "Character class"; any one character from those listed string

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = ".*a[bc]d.*";
        String Input1= "pqr abd xyz";
        String Input2 = "pqr acd xyz";

        //String Input = "abcabcdxyz";Output: not found

        boolean match1 = Pattern.matches(Regex, Input1);
        boolean match2 = Pattern.matches(Regex, Input2);
    }
}
```

```

        System.out.print("Output: ");
        if (match1)
            System.out.println(" found");
        else
            System.out.println(" not found");

        if (match2)
            System.out.println(" found");
        else
            System.out.println(" not found");
    }
}

```

Output: found

found

Example: Subexpression: [^\^...] Matches: Any one character not from those listed

```

import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String Regex = ".*a[b^c]d.*";
        String Input = "abc aed xyz";
        //String Regex = ".*ab[^c]d.*"; Output: not found
        boolean match = Pattern.matches(Regex, Input);
        System.out.print("Output: ");
        if (match)
            System.out.print(" found");
        else
            System.out.print(" not found");

    }
}

```

Output: found

Using regexes in Java: Test for a Pattern

- The `java.util.regex` package consists of two classes, `Pattern` and `Matcher` , which provide the public API shown as follows:

Example 4-1. Regex public API

```
/** The main public API of the java.util.regex package.
 * Prepared by javap and Ian Darwin.
 */
package java.util.regex;
public final class Pattern {
    // Flags values ('or' together)
    public static final int
        UNIX_LINES, CASE_INSENSITIVE, COMMENTS, MULTILINE,
        DOTALL, UNICODE_CASE, CANON_EQ;
    // No public constructors; use these Factory methods
    public static Pattern compile(String patt);
    public static Pattern compile(String patt, int flags);
    // Method to get a Matcher for this Pattern
    public Matcher matcher(CharSequence input);
    // Information methods
    public String pattern();
    public int flags();
    // Convenience methods
    public static boolean matches(String pattern, CharSequence input);
    public String[] split(CharSequence input);
    public String[] split(CharSequence input, int max);
}
```

```
public final class Matcher {
    // Action: find or match methods
    public boolean matches();
    public boolean find();
    public boolean find(int start);
    public boolean lookingAt();
    // "Information about the previous match" methods
    public int start();
    public int start(int whichGroup);
    public int end();
```

```

public int end(int whichGroup);
public int groupCount();
public String group();
public String group(int whichGroup);
// Reset methods
public Matcher reset();
public Matcher reset(CharSequence newInput);
// Replacement methods
public Matcher appendReplacement(StringBuffer where, String newText);
public StringBuffer appendTail(StringBuffer where);
public String replaceAll(String newText);
public String replaceFirst(String newText);
// information methods
public Pattern pattern();
}

```

```

/* String, showing only the RE-related methods */
public final class String {
    public boolean matches(String regex);
    public String replaceFirst(String regex, String newStr);
    public String replaceAll(String regex, String newStr);
    public String[] split(String regex);
    public String[] split(String regex, int max);
}

```

Matching regexes using Pattern and Matcher(s)

- If the regex is going to be used more than once or twice in a program, it is more efficient to construct and use a Pattern and its Matcher(s).
- *The normal steps for regex*
 1. Create a Pattern by calling the static method Pattern.compile().
 2. Request a Matcher from the pattern by calling pattern.matcher(CharSequence) for each String (or other CharSequence) you wish to look through.

3. Call (once or more) one of the finder methods (discussed later in this section) in the resulting Matcher.

The Matcher methods are:

matches()

Used to compare the entire string against the pattern; this is the same as the routine in `java.lang.String`. Because it matches the entire String, need to put `.*` before and after the pattern.

lookingAt()

Used to match the pattern only at the beginning of the string.

find()

Used to match the pattern in the string (not necessarily at the first character of the string), starting at the beginning of the string or, if the method was previously called and succeeded, at the first character not matched by the previous match.

Example: matches()

```
import java.util.regex.*;
```

```
public class RESimple {
```

public static void main(String[] args)

String pattern = ".*abd.*";

String input = "pqr bad pxy";

Pattern p = Pattern.compile(pattern);

```
Matcher m=p.matcher(input);
```

```
if(m.matches()) {
```

```
System.out.println("Patern " + pattern + "found in string " +  
                           input);
```

```

    }

Else {
    System.out.println("Patern " + pattern + "not found in string "
        + input);

}
}

}

```

Output: Patern .*abd.* found in string pqr abd pxy

Example: Another example on matches()

```

import java.util.regex.*;
public class RESimple {
    public static void main(String[] args) {
        String pattern = ".*Q[^u]\d+.*";
        String line = "Order QT300. Now!";
        if (line.matches(pattern)) {
            System.out.println(line + " matches \" " + pattern + " \"");
        }
        else {
            System.out.println("NO MATCH");
        }
    }
}

```

Output: Order QT300. Now! Matches ".*Q[^u]\d+.*"

Example: lookingAt()

```
import java.util.regex.*;
public class RESimple {
    public static void main(String[] args) {
        String pattern = "pqr";
        String input ="pqr abd pxy";
        Pattern p = Pattern.compile(pattern);
        Matcher m=p.matcher(input);
        if(m.lookingAt()){
            System.out.println("Patern "+pattern+
                " found in string "+input);
        }
        else {
            System.out.println("Patern "+pattern+
                " not found in string "+input);
        }
    }
}
```

Output: Patern pqr found in string pqr abd pxy

Example: Another example on lookingAt()

```
import java.util.regex.*;

public class RESimple {

    public static void main(String[] args) {

        String pattern = "Q[^u]\\d+\\.;" "Q[A-Za-z]\\d]+[\\.,,]"

        String[] input = { "QA777. is the next flight. It is on time.",
                           "Quack, Quack, Quack!" };

        Pattern p = Pattern.compile(pattern);

        for (String in : input) {

            boolean found = p.matcher(in).lookingAt();

            System.out.println("'" + pattern + "'"' +
                               (found ? " matches '" : " doesn't match") + in + "'");

        }
    }
}
```

Output: '^Q[^u]\\d+\\. matches 'QA777. is the next flight. It is on time.'
'^Q[^u]\\d+\\. doesn't match 'Quack, Quack, Quack!'

Example: find()

```
import java.util.regex.*;

public class RESimple {

    public static void main(String[] args) {

        String pattern = "abd";


```

```

String input ="pqr abd pxy";
Pattern p = Pattern.compile(pattern);
Matcher m=p.matcher(input);
if(m.find()) {
    System.out.println("Patern " + pattern +
                       "found in string "+input);
}
else {
    System.out.println("Patern " + pattern +
                       "not found in string "+input);
}
}
}

```

Output: Patern abd found in string pqr abd pxy

Example: Another example of find()

```

import java.util.regex.*;
public class RESimple {
    public static void main(String[] args) {
        String patt = ".*Q[^u]\d+.*";
        String line = "Order QT300. Now!";
        Matcher m = Pattern.compile(patt).matcher(line);
        if (m.find( )) {
            System.out.println(line + " matches " + patt);
        }
    }
}

```

```
    }  
  
    else {  
        System.out.println(line + "not matches " + patt);  
    }  
}
```

Output: Order QT300. Now! matches .*Q[^u]\d+\.*

Finding the Matching Text

- If you need to know exactly what characters were matched.
 - **start() and end()**
 - Returns the character position in the string of the starting and ending characters that matched.
 - The start() method returns the start index of first character of the subsequence(i.e., regex), matched into the string.
 - The end() method returns the index of the last character matched into the string, plus one.

Example:

```
import java.util.regex.*;  
  
public class Regex1 {  
  
    public static void main(String[] args) {  
  
        String pattern = "abd";  
        String input ="pqr abd pxy abd";  
        Pattern p = Pattern.compile(pattern);  
    }  
}
```

```

Matcher m=p.matcher(input);
int count=0;
while(m.find()) {
    count++;
    System.out.println("The " + count + " " + pattern + " Pattern
        found from " + m.start() + " to " + (m.end()-1));
}
}

```

Output:

The 1 abd Pattern found from 4 to 6
The 2 abd Pattern found from 12 to 14

groupCount()

- The `java.time.Matcher.groupCount()` method returns the number of capturing groups in this matcher's pattern.
- Returns the number of parenthesized capture groups, if any groups were used.
- Returns 0 if no groups were used.

Example:

```

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class GroupCount {

    public static void main(String[] args) {

        String regex = "(.*)(\d+)(.*)";
        String input = "This is a sample Text, 1234, with numbers in between.';

        //Creating a pattern object
        Pattern pattern = Pattern.compile(regex);
    }
}

```

```

//Matching the compiled pattern in the String
Matcher matcher = pattern.matcher(input);

System.out.println("Number of groups capturing: "
+ matcher.groupCount());

}

}

```

Output:

Number of groups capturing: 3

Example: Another Example of groupCount()

```

import java.util.regex.*;

public class Regex1 {

public static void main(String[] args) {

    //String pattern = ".*\\d{6}"; //Output: Total group = 0
    //String pattern = "(.*)((\\d{6}))"; //Output: Total group = 2
    String pattern = "(.*)((\\d{6}))(.*>"; //Output: Total group = 3

    String input ="abdpaxy 100000";

    Pattern p = Pattern.compile(pattern);
    Matcher m=p.matcher(input);

    System.out.println("Total group = "+m.groupCount());
}
}

```

Output:

Total group = 3

group(int i)

- The group(int i) method retrieve the characters that matched a given parenthesis group.
- Returns the characters matched by group i of the current match, if i is greater than or equal to zero and less than or equal to the return value of groupCount() .
- Group 0 is the entire match, so group(0) (or just group()) returns the entire portion of the input that matched.
- If you haven't used any explicit parens, you can just treat whatever matched as "level zero."

Example:

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class GroupCount {

    public static void main(String[] args) {

        String regex = "(.*)(\\d+)(.*";
        String input = "This is a sample Text, 1234, with numbers in between.';

        //Creating a pattern object
        Pattern pattern = Pattern.compile(regex);

        //Matching the compiled pattern in the String
        Matcher matcher = pattern.matcher(input);

        if(matcher.find()) {
            System.out.println("First group match: "+matcher.group(1));
            System.out.println("Second group match: "+matcher.group(2));
            System.out.println("Third group match: "+matcher.group(3));
            System.out.println("Number of groups capturing: "
                + matcher.groupCount());
        }
    }
}
```

```
        }
    }
}
```

Output:

First group match: This is a sample Text, 123

Second group match: 4

Third group match: , with numbers in between.

Number of groups capturing: 3

Example:

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String pattern = "(.*)(\\d{6})(.*)"; //Output: Total group = 3
        String input ="abdpwy 100000";

        Pattern p = Pattern.compile(pattern);
        Matcher m=p.matcher(input);

        int k = m.groupCount();
        System.out.println("Total group count= "+ k);

        while(m.find()) {
            for(int i=0; i<=k;i++)
                System.out.println("Patern "+pattern+" found in string "
                    + input +" with group "+ i + " is " +
                    m.group(i));

        }
    }
}
```

Output:

Total group count= 3

Patern (.*)\\d{6}(.*) found in string abdpxy 100000 with group 0 is abdpxy
100000

Patern (.*)\\d{6}(.*) found in string abdpxy 100000 with group 1 is abdpxy

Patern (.*)\\d{6}(.*) found in string abdpxy 100000 with group 2 is 100000

Patern (.*)\\d{6}(.*) found in string abdpxy 100000 with group 3 is

Example: Write a java program to display the formatted phone number.

```
import java.util.regex.*;  
  
public class Regex1 {  
  
    public static void main(String[] args) {  
  
        String regex = "\\b(\\d{3})(\\d{3})(\\d{4})\\b";  
        Pattern p = Pattern.compile(regex);  
  
        String source = "1234567890, 12345, and 9876543210";  
        Matcher m = p.matcher(source);  
  
        int k = m.groupCount();  
        System.out.println("Total group count= "+ k);  
  
        while (m.find()) {  
            System.out.println("Phone: " + m.group() + ",  
                Formatted Phone: (" + m.group(1) + ") " + m.group(2) + "-" +  
                m.group(3));  
        }  
    }  
}
```

Output:

Total group count= 3

Phone: 1234567890, Formatted Phone: (123) 456-7890

Phone: 9876543210, Formatted Phone: (987) 654-3210

Example: If the input is: *Adams, John Quincy*
and want to get output: ***John Quincy Adams***

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String inputLine = "Adams, John Quincy";
        Pattern r = Pattern.compile("(.*), (.*)");
        Matcher m = r.matcher(inputLine);

        if (m.matches())
            System.out.println(m.group(2) + ' ' + m.group(1));
    }
}
```

Output:

John Quincy Adams

Replacing the Matched Text

- Replace the text that the regex matched without changing the other text before or after it.
- **replaceAll(newString)**
 - Replaces all occurrences that matched with the new string.

Example:

```
import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {

        String patt = "\bfavor\b";
    }
}
```

```

String input = "Do me a favor? Fetch my favorite. favor me.";
System.out.println("Input: " + input);

Pattern r = Pattern.compile(patt);
Matcher m = r.matcher(input);
System.out.println("After Replace: " + m.replaceAll("help"));
}

}

```

Output:

Input: Do me a favor? Fetch my favorite. favor me.
After Replace: Do me a help? Fetch my favorite. help me.

Another replacement technique is as follows:

- **appendReplacement(StringBuffer, newString)**
 - Copies up to before the first match, plus the given newString.
- **appendTail(StringBuffer)**
 - Appends text after the last match (normally used after *appendReplacement*).

Example:

```

import java.util.regex.*;

public class Regex1 {

    public static void main(String[] args) {
        String patt = "favor";
        String input = "Do me a favor? Fetch my favorite. favor me.";
        System.out.println("Input: " + input);

        Pattern r = Pattern.compile(patt);
        Matcher m = r.matcher(input);

```

```

StringBuffer sb = new StringBuffer();
System.out.print("Append methods: ");

while (m.find()) {
    // Copy to before first match, plus the word "help"
    m.appendReplacement(sb, "help");
}
m.appendTail(sb); // copy remainder
System.out.println(sb.toString());
}
}

```

Output:

Input: Do me a favor? Fetch my favorite. favor me.

Append methods: Do me a help? Fetch my favorite. help me.

Java Pattern compile() Method

- The compile() method of Pattern class is used to compile the given regular expression passed as the string.
- It used to match text or expression against a regular expression more than one time.
- Syntax:
 - Pattern.compile(String regex)
 - Return compiled version of a regular expression into the pattern.
 - Pattern.compile(String regex, flag)
 - Return compiled version of a regular expression into the pattern with the given flag.
 - Here
 - regex - The expression that we want to compile.

flags - Match flags, a bit mask that may includes, CASE_INSENSITIVE, MULTILINE, DOTALL, UNICODE_CASE, CANON_EQ, UNIX_LINES, LITERAL, UNICODE_CHARACTER_CLASS and COMMENTS.

- If more than one value is needed, they can be added together using the bitwise or operator | .

CANON_EQ

- Enables so-called “canonical equivalence.” In other words, characters are matched by their base character, so that the character e followed by the “combining character mark” for the acute accent (́) can be matched either by the composite character é or the letter e followed by the character mark for the accent.

Example:

```
import java.util.regex.*;

public class PatternCompileFlags {

    public static void main(String[] args) {

        String pattStr = "\u00e9gal";

        String[] input = { "\u00e9gal", "e\u0301gal", "e\u02cagal",
                           "e'gal","e\u00b4gal" };

        System.out.println("Pattern: "+ pattStr);
        System.out.print("Input: ");

        for (int i = 0; i < input.length; i++){

            System.out.print(" " + input[i]);
        }
    }
}
```

```

}

System.out.println(" ");

Pattern pattern=Pattern.compile(pattStr,
                                Pattern.CANON_EQ);

for (int i = 0; i < input.length; i++){

    if(pattern.matcher(input[i]).matches()){

        System.out.println(pattStr + "matches input
                           " + input[i]);

    }

    else{

        System.out.println(pattStr + "does not match
                           input " + input[i]);

    }

}
}

```

Output:

Pattern: égal

Input: égal égal e'gal e'gal e'gal

égal matches input égal

égal matches input égal

égal does not match input e'gal

égal does not match input e'gal

égal does not match input e'gal

CASE_INSENSITIVE and UNICODE_CASE

- **CASE_INSENSITIVE:** Compile the Pattern passing in the flags argument Pattern.CASE_INSENSITIVE to indicate that matching should be case-independent (ignore differences in case).
- **UNICODE_CASE:** Enables Unicode-aware case folding. If your code might run in different locales then you should add Pattern.UNICODE_CASE.

Example:

```
import java.util.regex.*;

public class PatternCompileFlags {

    public static void main(String[] args) {

        String regex = "Pqr";
        String input = "abc Pqr xyz pqr";
        Pattern p=Pattern.compile(regex, Pattern.CASE_INSENSITIVE
                           | Pattern.UNICODE_CASE);

        Matcher m=p.matcher(input);

        While (m.find()) {

            System.out.println("The " + p + "      Pattern found from " +
                               m.start() + " to " + (m.end()-1));
        }
    }
}
```

Output:

The Pqr Pattern found from 4 to 6
The Pqr Pattern found from 12 to 14

MULTILINE, UNIX_LINES and DOTALL

- **MULTILINE**
 - Specifies multiline mode, which makes newlines match as beginning-of-line and end-of-line (^ and \$).
- **DOTALL**
 - Allows dot (.) to match any regular character or the newline, not just any regular character other than newline.
- **UNIX_LINES**
 - Makes \n the only valid “newline” sequence for MULTILINE mode

Example:

```
import java.util.regex.*;  
  
public class PatternCompileFlags {  
  
    public static void main(String[] args) {  
  
        String input = "I dream of engines\nmore engines, all day long";  
        System.out.println("INPUT: " + input);  
  
        System.out.println();  
  
        String[] patt = {"engines.more engines", "ines\nmore", "engines$"};  
  
        System.out.println("PATTERN: ");  
  
        for (int i = 0; i < patt.length; i++) {  
  
            System.out.println(patt[i]);  
  
        }  
  
        System.out.println();  
  
        for (int i = 0; i < patt.length; i++) {  
  
            System.out.println("PATTERN: " + patt[i]);  
  
        }  
    }  
}
```

```

boolean found;

Pattern p11 = Pattern.compile(patt[i]);
found = p11.matcher(input).find();
System.out.println("DEFAULT match: " + found);

Pattern pml = Pattern.compile(patt[i], Pattern.DOTALL |
                                Pattern.MULTILINE);
//Pattern pml = Pattern.compile(patt[i], Pattern.DOTALL |
                                Pattern.UNIX_LINES);

found = pml.matcher(input).find();
System.out.println("MultiLine match: " + found);
System.out.println();
}

}

```

Output:

INPUT: I dream of engines
more engines, all day long

PATTERN:

engines.more engines
ines
more
engines\$

PATTERN: engines.more engines

DEFAULT match: false

MultiLine match: true

PATTERN: ines

more

DEFAULT match: true

MultiLine match: true

PATTERN: engines\$

DEFAULT match: false

MultiLine match: true

COMMENTS

- Causes whitespace and comments (from # to end-of-line) to be ignored in the pattern.

Example:

```
import java.util.regex.*;

public class PatternCompileFlags {

    public static void main(String[] args) {

        String regex = "Pqr #HI I AM HERE./";

        Pattern p =
            Pattern.compile(regex, Pattern.CASE_INSENSITIVE
                | Pattern.COMMENTS);

        String input = "abc Pqr xyz pqr";
        Matcher m=p.matcher(input);
```

```

while(m.find()) {
    System.out.println("The " + p + "      Pattern found from "
                      + m.start() + " to " + (m.end()-1));
}
}

```

Output:

The Pqr #HI I AM HERE. Pattern found from 4 to 6
The Pqr #HI I AM HERE. Pattern found from 12 to 14

1 Write a program in Java to remove whitespaces from a string.

```

public class RegexExercise {

    public static void main(String[] args) {
        String str = "ITER Is My College";
        System.out.println("Input String: " + str);
        System.out.println();
        //1st way (using built in method)
        String str1 = str.replaceAll("\\s", "");
        System.out.println("String After Remove Spacce: " + str1);
        System.out.println();
        //2nd way // using StringBufer() method
        char[] strArray = str.toCharArray();
        StringBuffer stringBuffer = new StringBuffer();

```

```

for (int i = 0; i < strArray.length; i++) {
    if ((strArray[i] != ' ') && (strArray[i] != '\t'))
    {
        stringBuffer.append(strArray[i]);
    }
}

String str2 = stringBuffer.toString();
System.out.println("String After Remove Spacce: " + str2);
}
}

```

Output:

Input String: ITER Is My College

String After Remove Spacce: ITERIsMyCollege

String After Remove Spacce: ITERIsMyCollege

2 Write a program in Java to check valid mobile number.

Mobile Number validation criteria:

- The first digit should contain number between 7 to 9.
- The rest 9 digit can contain any number between 0 to 9.
- The mobile number can have 11 digits also by including 0 at the starting.

- The mobile number can be of 12 digits also by including 91 at the starting

The number which satisfies the above criteria, is a valid mobile Number.

```
import java.util.regex.*;  
  
class MobileNumberValidation {  
  
    public static void main(String[] args) {  
  
        String regex = "(0|(91))?[789][0-9]{9}";  
        Pattern ptr = Pattern.compile(regex);  
        String mobileno = "9439342133"; //valid.  
        //String mobileno = "09439342133"; //valid.  
        //String mobileno = "919439342133"; //valid.  
        //String mobileno = "913439342133"; //not valid.  
        Matcher m = ptr.matcher(mobileno);  
        if(m.matches())  
        {  
            System.out.println("The Mobile number " + mobileno +  
                " is valid.");  
        }  
        else  
        {  
            System.out.println("The Mobile number " + mobileno +  
                " is not valid.");  
        }  
    }  
}
```

Output :

The Mobile number 9439342133 is valid.

3 Write a program in Java to check valid email

```
import java.util.regex.*;
public class RegexExercise {
    public static void main(String[] args) {
        String emailRegex = "^(a-zA-Z0-9_\\|-\\|.)"
        +")@([a-zA-Z0-9_\\|-\\|.]*)\\.(a-zA-Z){2,5}$";
        Pattern pat = Pattern.compile(emailRegex);
        String email = "contribute@gmail.co.in";
        if (pat.matcher(email).matches())
            System.out.print("Yes");
        else
            System.out.print("No");
    }
}
```

Output:

yes

4. Write a program to extract maximum numeric value from a given string. For example: this is aIJThere is 60 students in csed section, 40 in cseb, and 55 in cseaâI.

```
import java.util.regex.*;
public class RegexExercise {
```

```

public static void main(String[] args) {

    String str = "this is aIJThere is 60 students in csed section,
                 40 in cseb, and 55 in cseaâ€œ";
    String regex = "\\d+";
    Pattern p = Pattern.compile(regex);
    Matcher m = p.matcher(str);
    int MAX = 0;
    while(m.find()) {

        int num = Integer.parseInt(m.group());
        if(num > MAX)
            MAX = num;
    }
    System.out.println(MAX);
}
}

```

Output:

60

5 Write a program to demonstrate the working of splitting a text by a given pattern. The given input is “CSE1ECE2EEE3CIVIL”.The output of the program is look like below:

CSE

ECE

EEE

CIVIL

Use the split () and case controlling flags to solve this.

```
import java.util.regex.*;

public class RegexExercise {

    public static void main(String[] args) {

        String s1 = "CSE1ECE2EEE3CIVIL";
        String regex1 = "\d";

        String[] arrOfStr = s1.split(regex1, 5);

        for (String a : arrOfStr)
            System.out.println(a);
    }
}
```

Output:

```
CSE
ECE
EEE
CIVIL
```

6 Write a program to get the first letter of each word in a string using regex in Java. For example: the input string is “This is CSE Students” and output of the program is: TiCS.

```
import java.util.regex.*;

public class RegexExercise {

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

```

String s1 = "This is CSE Students";
Pattern p = Pattern.compile("\\b[a-zA-Z]");
Matcher m = p.matcher(s1);
while (m.find())
    System.out.print(m.group());
}

```

Output:

TiCS

7 Write a program to demonstrate the differences of various quantifiers e.g. Greedy Quantifiers VS Reluctant Quantifiers VS Possessive Quantifiers.

Regular Expressions in Java

Quantifiers allow user to specify the number of occurrences to match against.

Below are some commonly used quantifiers in Java.

X* Zero or more occurrences of X

X? Zero or One occurrences of X

X+ One or More occurrences of X

X{n} Exactly n occurrences of X

X{n, } At-least n occurrences of X

X{n, m} Count of occurrences of X is from n to m

The above quantifiers can be made Greedy, Reluctant and Possessive.

Greedy quantifier (Default)

By default, quantifiers are Greedy. Greedy quantifiers try to match the longest text that matches given pattern. Greedy quantifiers work by first reading the entire string before trying any match. If the entire text doesn't match, remove last character and try again, repeating the process until a match is found.

Output :

Pattern found from 0 to 2

Explanation : The pattern **g+** means one or more occurrences of **g**. Text is **ggg**.

The greedy matcher would match the longest text even if parts of matching text also match. In this example, **g** and **gg** also match, but the greedy matcher produces **ggg**.

Reluctant quantifier (Appending a ? after quantifier)

This quantifier uses the approach that is opposite of greedy quantifiers. It starts from first character and processes one character at a time.

```
// Java program to demonstrate Reluctant Quantifiers

import java.util.regex.Matcher;
import java.util.regex.Pattern;

class Test {

    public static void main(String[] args)

    {
        // Making an instance of Pattern class. Here "+" is a Reluctant
        // quantifier because a "?" is appended after it.

        Pattern p = Pattern.compile("g+?");

        // Making an instance of Matcher class

        Matcher m = p.matcher("ggg");

        while (m.find())
```

```
        System.out.println("Pattern found from " + m.start() + " to " +
                           (m.end()-1));
    }
}
```

Output :

Pattern found from 0 to 0

Pattern found from 1 to 1

Pattern found from 2 to 2

Explanation : Since the quantifier is reluctant, it matches the shortest part of test with pattern. It processes one character at a time.

Possessive quantifier (Appending a + after quantifier)

This quantifier matches as many characters as it can like greedy quantifier. But if the entire string doesn't match, then it doesn't try removing characters from end.

```
// Java program to demonstrate Possessive Quantifiers
```

```
import java.util.regex.Matcher;
```

```
import java.util.regex.Pattern;
```

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```

// Making an instance of Pattern class. Here "+" is a Possessive quantifier
because a '+' is appended after it.

Pattern p = Pattern.compile("g++");

// Making an instance of Matcher class

Matcher m = p.matcher("ggg");

while (m.find())

    System.out.println("Pattern found from " + m.start() + " to " +
                       (m.end()-1));

}

```

Output :

Pattern found from 0 to 2

Explanation: We get the same output as Greedy because whole text matches the pattern.

Below is an example to show difference between Greedy and Possessive Quantifiers.

```

// Java program to demonstrate difference between Possessive and
// Greedy Quantifiers

import java.util.regex.Matcher;

import java.util.regex.Pattern;

class Test {

    public static void main(String[] args)

    {

```

```

// Create a pattern with Greedy quantifier

Pattern pg = Pattern.compile("g+g");

// Create same pattern with possessive quantifier

Pattern pp = Pattern.compile("g++g");

System.out.println("Using Greedy Quantifier");

Matcher mg = pg.matcher("ggg");

while (mg.find())

    System.out.println("Pattern found from " + mg.start() + " to "
                       + (mg.end()-1));

System.out.println("\nUsing Possessive Quantifier");

Matcher mp = pp.matcher("ggg");

while (mp.find())

    System.out.println("Pattern found from " + mp.start() + " to "
                       + (mp.end()-1));

}

```

Output :

Using Greedy Quantifier

Pattern found from 0 to 2

Using Possessive Quantifier

In the above example, since first quantifier is greedy, **g+** matches with whole string. If we match **g+** with whole string, **g+g** doesn't match, the Greedy quantifier, removes last character, matches **gg** with **g+** and finds a match.

In Possessive quantifier, we start like Greedy. **g+** matches whole string, but matching **g+** with whole string doesn't match **g+g** with **ggg**. Unlike Greedy, since quantifier is possessive, we stop at this point.

NUMBER

- Numbers are basic to just about any computation.
- Furthermore, the numbers are used for array indices, temperatures, salaries, ratings, and an infinite variety of things.
- Java has represent numbers as follows:
 - built-in or primitive types
 - wrapper (object) types

Built-in type	Object wrapper	Size of built-in (bits)	Contents
byte	Byte	8	Signed integer
short	Short	16	Signed integer
int	Integer	32	Signed integer
long	Long	64	Signed integer
float	Float	32	IEEE-754 floating point
double	Double	64	IEEE-754 floating point
n/a	BigInteger	UnlimitedArbitrary-size	immutable integer value
n/a	BigDecimal	Unlimited Arbitrary-size and-precision	immutable floating-point value

Wrapper Classes in Java

- A Wrapper class is a class whose object wraps or contains a primitive data types.
- When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types.
- In other words, we can wrap a primitive value into a wrapper class object.

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive.*
 - **Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.
 - **Unboxing:** It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Need of Wrapper Classes

- **Change the value in Method:** Java supports only call by value. So, if we pass a primitive value, it will not change the original value. Hence, convert primitive data types into objects through the wrapper classes. Objects are needed if we wish to modify the arguments (i.e., original value) passed into a method.
- **Serialization:** We need to convert the objects into streams to perform the serialization.
 - A *stream* is a sequence of objects that supports various methods which can be pipelined to produce the desired result.
- **Synchronization:** Java synchronization works with objects in Multithreading.
- **java.util package:** The java.util package provides the utility classes to deal with objects.
- **Collection Framework:** Java collection framework works with objects only. All classes of the collection framework (ArrayList, LinkedList, Vector, HashSet, Linked HashSet, TreeSet, Priority Queue, Array Deque, etc.) deal (i.e., store) only with objects (reference types) and not primitive types.


```

Long L1 = new Long(40);           //Constructor which takes
                                  long value as an argument
Long L2 = new Long("40");         //Constructor which takes
                                  String as an argument

Float F1 = new Float(12.2f);      //Constructor which takes
                                  float value as an argument
Float F2 = new Float("15.6");     //Constructor which takes
                                  String as an argument
Float F3 = new Float(15.6d);      //Constructor which takes
                                  double value as an argument

Double D1 = new Double(17.8d);    //Constructor which takes
                                  double value as an argument
Double D2 = new Double("17.8");   //Constructor which takes
                                  String as an argument

Boolean BLN1 = new Boolean(false); //Constructor which takes
                                  boolean value as an argument
Boolean BLN2 = new Boolean("true"); //Constructor which takes
                                  String as an argument

Character C1 = new Character('D'); //Constructor which takes
                                  char value as an argument
Character C2 = new Character("abc"); //Compile time error :
                                  String abc can not be
                                  converted to character
}

}

```

Parsing Methods Of Wrapper Classes In Java

- All wrapper classes in java have methods to parse the given string to corresponding primitive data provided the string should be parse-able.
- If the string is not parse-able, you will get NumberFormatException.
- All parsing methods of wrapper classes are static i.e., you can refer them directly using class name.

Example:

```
public class Number {  
    public static void main(String[] args) {  
  
        byte b = Byte.parseByte("10");  
        System.out.println(b); //Output : 10  
  
        short s = Short.parseShort("25");  
        System.out.println(s); //Output : 25  
  
        int i = Integer.parseInt("123");  
        System.out.println(i); //Output : 123  
  
        long l = Long.parseLong("100");  
        System.out.println(l); //Output : 100  
  
        float f = Float.parseFloat("12.35");  
        System.out.println(f); //Output : 12.35  
  
        double d = Double.parseDouble("12.87");  
        System.out.println(d); //Output : 12.87  
  
        boolean bln = Boolean.parseBoolean("true");  
        System.out.println(bln); //Output : true  
  
        boolean bln1 = Boolean.parseBoolean("abc");  
        System.out.println(bln1); //Output : false  
  
        //char c = Character.parseChar("abc"); //compile time error  
        //parseChar() is not defined for Character wrapper class  
    }  
}
```

Output:

10
25
123
100
12.35
12.87
true
false

Methods of wrapper class

- The most common methods of the Integer wrapper class are summarized in below table. Similar methods for the other wrapper classes are found in the Java API documentation.

Method	Purpose
<code>parseInt(s)</code>	returns a signed decimal integer value equivalent to string s
<code>toString(i)</code>	returns a new String object representing the integer i
<code>byteValue()</code>	returns the value of this Integer as a byte
<code>doubleValue()</code>	returns the value of this Integer as a double
<code>floatValue()</code>	returns the value of this Integer as a float
<code>intValue()</code>	returns the value of this Integer as an int
<code>shortValue()</code>	returns the value of this Integer as a short
<code>longValue()</code>	returns the value of this Integer as a long
<code>int compareTo(int i)</code>	Compares the numerical value of the invoking object with that of i. Returns 0 if the values are equal. Returns a negative value if the invoking object has a lower value. Returns a positive value if the invoking object has a greater value.
<code>static int compare(int num1, int num2)</code>	Compares the values of num1 and num2. Returns 0 if the values are equal. Returns a negative value if num1 is less than num2. Returns a positive value if num1 is greater than num2.
<code>boolean equals(Object, intObj)</code>	Returns true if the invoking Integer object is equivalent to intObj. Otherwise, it returns false.

Example:

```
public class Number {  
  
    public static void main(String[] args) {  
  
        Integer intObj1 = new Integer (25);  
        Integer intObj2 = new Integer ("25");  
        Integer intObj3= new Integer (35);  
    }  
}
```

```

//compareTo demo
System.out.println("Comparing by using compareTo Obj1
and Obj2: " + intObj1.compareTo(intObj2));
System.out.println("Comparing by using compareTo Obj1
and Obj3: " + intObj1.compareTo(intObj3));

//Equals demo
System.out.println("Comparing by using equals Obj1 and
Obj2: " + intObj1.equals(intObj2));
System.out.println("Comparing by using equals Obj1 and
Obj3: " + intObj1.equals(intObj3));

Float f1 = new Float("2.25f");
Float f2 = new Float("20.43f");
Float f3 = new Float(2.25f);
System.out.println("Comparing by using compare f1 and f2: "
+ Float.compare(f1,f2));
System.out.println("Comparing by using compare f1 and f3: "
+ Float.compare(f1,f3));

//Addition of Integer with Float
Float f = intObj1.floatValue() + f1;
System.out.println("Addition of intObj1 and f1: " +
intObj1 +"+" +f1+"=" +f );
}

}

```

Output:

Comparing using compareTo Obj1 and Obj2: 0
Comparing using compareTo Obj1 and Obj3: -1
Comparing using equals Obj1 and Obj2: true
Comparing using equals Obj1 and Obj3: false
Comparing using compare f1 and f2: -1
Comparing using compare f1 and f3: 0
Addition of intObj1 and f1: 25+2.25=27.25

Converting Numbers to Objects and Vice Versa

- The **wrapper class in Java** provides the mechanism *to convert primitive into object and object into primitive.*
 - **Autoboxing:** Automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double etc.
 - **Unboxing:** It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double etc.

Example: Auto-boxing and Auto-unboxing

```
public class Number {  
  
    public static void main(String[] args) {  
  
        System.out.println("Converting Primitive type to Object type");  
        Integer IObj = 10; //Auto-boxing  
        System.out.println(IObj);  
        Float FObj = 10.6f; //Auto-boxing  
        System.out.println(FObj);  
        Double DObj = 12.5; //Auto-boxing  
        System.out.println(DObj);  
  
        System.out.println();  
  
        System.out.println("Converting Object type to Primitive type");  
        int iObj = IObj; //Auto-unboxing  
        System.out.println(iObj);  
        float fObj = FObj; //Auto-unboxing  
        System.out.println(fObj);  
        double dObj = DObj; //Auto-unboxing  
        System.out.println(dObj);  
  
    }  
}
```

Output:

Converting Primitive type to Object type

10

10.6

12.5

Converting Object type to Primitive type

10

10.6

12.5

Example: Explicitly convert between an int and an Integer object, or vice versa, by using the wrapper class methods.

```
public class Number {  
  
    public static void main(String[] args) {  
  
        // int to Integer  
        // Integer i1 = new Integer(42);  
        Integer i1 = Integer.valueOf(42);  
        System.out.println(i1.toString()); // or just i1  
  
        // Integer to int  
        int i2 = i1.intValue();  
        System.out.println(i2);  
    }  
}
```

Output:

42

42

Example: Shows autoboxing (in the call to foo(i), i is wrapped automatically) and auto-unboxing (the return value is automatically unwrapped).

```
public class Number {  
  
    public static void main(String[] args) {  
  
        int i = 42;  
    }  
}
```

```
int result = foo(i);                                //Auto-unboxing
System.out.println(result);
}

public static Integer foo(Integer i) {                //Auto-boxing
    System.out.println("Object = " + i);
    Integer sb = new Integer(123);
    // Integer sb = Integer.valueOf(123)
    return sb;
}
}
```

Output:

Object = 42
123

Checking Whether a String Is a Valid Number

Example: Write a program which read a sting and convert it to integer number.

```
import java.util.Scanner;
```

public class Number {

```
public static void main(String[] args) {  
  
    Scanner sc = new Scanner(System.in);  
    System.out.println("Enter a number");  
    String aNumber = sc.next();  
    int result;  
  
    try {  
        result = Integer.parseInt(aNumber);  
        System.out.println("The " + result + " is a valid integer.");  
    }  
    catch(NumberFormatException exc) {  
  
        System.out.println("The " + aNumber + " is an invalid  
                           integer.");  
    }  
    return;  
}
```

```
    }  
}
```

Output:

Enter a number

12345

The 12345 is a valid integer.

Enter a number

1234.67

The 1234.67 is an invalid integer.

Example: Write a program which read a string and convert it to double number.

```
import java.util.Scanner;
```

```
public class Number {
```

```
    public static void main(String[] args) {  
  
        Scanner sc=new Scanner(System.in);  
        System.out.println("Enter a number");  
        String aNumber = sc.next();  
        double result;  
  
        try {  
            result = Double.parseDouble(aNumber);  
            System.out.println("The " + result + " is a valid double.");  
        }  
        catch(NumberFormatException exc) {  
  
            System.out.println("The " + aNumber + " is an invalid  
double.");  
            return;  
        }  
    }  
}
```

Output:

Enter a number

1234.67

The 1234.67 is a valid double.

Storing a Larger Number in a Smaller Number

Create a float variable and assign it 3.0 to it and observe what happens.

```
float f = 3.0;
```

if you had written:

```
double tmp = 3.0;  
float f = tmp; //won't even compile!
```

It can be fixed by any one of several ways:

- 1) By making the 3.0 a float (probably the best solution)

```
float f = 3.0f; // or just 3f
```

- 2) By making f a double

```
double f = 3.0;  
float f = (float)3.0;// By putting in a cast
```

- 3) By assigning an integer value of 3, which will get "promoted"

```
float f = 3;
```

Example: Write a program which converts double number to integer and byte type.

```
public class Number {  
  
    public static void main(String[] args) {  
  
        int i;  
        double j = 2.75;  
        //i = j; // EXPECT COMPILE ERROR  
        i = (int) j; // with cast; i gets 2  
        System.out.println("i =" + i);  
  
        byte b;  
        //b = i; // EXPECT COMPILE ERROR  
        b = (byte) i; // with cast, i gets 2  
        System.out.println("b =" + b);  
    }  
}
```

Output:

i = 2
b = 2

Taking a Fraction of an Integer Without Using Floating Point

suppose you want to multiply, **0.666 * 5** then

- multiply an integer by a fraction without converting the fraction to a floating-point number.
- Multiply the integer by the numerator and divide by the denominator.
- integers and floating-point numbers are stored differently.
- for efficiency purposes, to multiply an integer by a fractional value without converting the values to floating point and back.
- it doesn't require a “cast”.

Example:

Public class Number {

```
Public static void main(String[] args) {  
  
    double d1 = 0.666 * 5;           //fast but obscure and inaccurate  
    System.out.println(d1);  
  
    double d2 = 2/3 * 5;           // convert 0.666 to 2/3  
    System.out.println(d2);           //wrong answer, 2/3 = 0, 0*5 = 0  
  
    double d3 = 2d/3d * 5;         // "normal"  
    System.out.println(d3);  
  
    double d4 = (2*5)/3d;          // one step done as integers,  
    System.out.println(d4);           //almost same answer  
  
    int i5 = 2*5/3;                // fast, approximate integer answer  
    System.out.println(i5);
```

```
    }  
}
```

Output:

```
3.3  
0.0  
3.33333333333333  
3.33333333333335  
3
```

Ensuring the Accuracy of Floating-Point Numbers

- In java, integer division by 0 consider as logical error so it throws an ArithmeticException.
- floating-point computation generated a sensible result
- Floating-point operations, however, do not throw an exception because they are defined over an (almost) infinite range of values.
 - if you divide a positive floating-point number by zero, then Java signal errors by producing the constant POSITIVE_INFINITY.
 - if you divide a negative floating-point value by zero, then Java signal errors by producing the constant NEGATIVE_INFINITY.
 - Otherwise generate an invalid result NaN (Not a Number)
- Values for these three public constants are defined in both the Float and the Double wrapper classes.
- The value NaN has the unusual property that it is not equal to itself (i.e., `NaN != NaN`).
- `x==NaN` never be true,
- To check particular value is NaN or not, the methods `Float.isNaN(float)` and `Double.isNaN(double)` must be used.

Example:

```
public class Number {  
  
    public static void main(String[] args) {  
  
        double d = 123;  
        double e = 0;  
        if (d/e == Double.POSITIVE_INFINITY)  
            System.out.println("Check for POSITIVE_INFINITE");  
  
        double s = Math.sqrt(-1);  
        System.out.println("The s = " + s);  
  
        if (s == Double.NaN)  
            System.out.println("Comparison with NaN incorrectly  
                                returns true");  
        if (Double.isNaN(s))  
            System.out.println("Double.isNaN() correctly returns true");  
    }  
}
```

Output:

Check for POSITIVE_INFINITY works

The s = NaN

Double.isNaN() correctly returns true

Example: Calculate the area of a triangle by using Heron's formula for both in float and in double.

```
public class Number {  
  
    public static void main(String[] args) {  
  
        float af, bf, cf;  
        float sf, areaf;  
  
        double ad, bd, cd;  
        double sd, aread;  
  
        // Area of triangle in float  
        af = 12345679.0f;
```

```

bf = 12345678.0f;
cf = 1.01233995f;

sf = (af+bf+cf)/2.0f;
System.out.println("sf = " + sf);
areaf = (float)Math.sqrt(sf * (sf - af) * (sf - bf) * (sf - cf));
System.out.println("Single precision: " + areaf);

// Area of triangle in double
ad = 12345679.0;
bd = 12345678.0;
cd = 1.01233995;

sd = (ad+bd+cd)/2.0d;
System.out.println("sd = " + sf);
aread = Math.sqrt(sd * (sd - ad) * (sd - bd) * (sd - cd));
System.out.println("Double precision: " + aread);
}

}

```

Output:

sf = 1.2345679E7	
Single precision: 0.0	//result is incorrect
sd = 1.2345679E7	
Double precision: 972730.0557076167	//result is correct

The double values are correct, but the floating-point value comes out as zero due to rounding errors. This happens because, in Java, operations involving only float values are performed as 32-bit calculations. Related languages such as C automatically promote these to double during the computation, which can eliminate some loss of accuracy.

Comparing Floating-Point Numbers

- Compare two floating-point numbers for equality.
 - The equals() method of Float and Double class returns true if the two values are the same bit for bit (i.e., if and only if the numbers are the same or are both NaN).

- To actually compare floating-point numbers for equality, it is generally desirable to compare them within some tiny range of allowable differences; this range is often regarded as a tolerance or as epsilon.

Example:

```

public class Number {

    final static double EPSILON = 0.0000001;

    /** Compare two doubles within a given epsilon */
    Public static boolean equals(double a, double b, double eps) {
        if (a==b)
            return true;
        // If the difference is less than epsilon, treat as equal.
        return Math.abs(a - b) < eps;
    }

    /** Compare two doubles, using default epsilon */
    public static boolean equals(double a, double b) {
        return equals(a, b, EPSILON);
    }

    public static void main(String[] args) {

        double da = 3 * .3333333333;
        double db = 0.99999992857;

        // Compare two numbers that are expected to be close.
        if (da == db) {
            System.out.println("Java considers " + da + "==" + db);
        }

        // else compare with our own equals overload
        elseif (equals(da, db, EPSILON)) {
            System.out.println("Equal within epsilon " + EPSILON);
        }
        else {
            System.out.println(da + " != " + db);
        }
    }
}

```

Output:

Equal within epsilon 1.0E-7

Rounding Floating-Point Numbers

- To round floating-point numbers properly, use Math.round().
- It has two overloads:
 - if you give it a double, you get a long type result.
 - if you give it a float, you get an int type result.

Example:

```
public class Number {  
  
    public static void main(String[] args) {  
  
        double d=5.67;  
        System.out.println("The rounding of " + d + " is " + Math.round(d));  
  
        float f=9.4255f;  
        System.out.println("The rounding of " + f + " is " + Math.round(f));  
    }  
}
```

Output:

The rounding of 5.67 is 6

The rounding of 9.4255 is 9

Example: Rounding the number from 0.1, 0.15, 0.2, 0.25,, 0.95.

```
public class Number {  
  
    public static void main(String[] args) {  
  
        for (double d = 0.1; d<1.0; d+=0.05) {  
  
            System.out.println("The rounding of " + d + " is " + Math.round(d));  
        }  
    }  
}
```

```
    }  
}
```

Output:

```
The rounding of 0.1 is 0  
The rounding of 0.1500000000000002 is 0  
The rounding of 0.2 is 0  
The rounding of 0.25 is 0  
The rounding of 0.3 is 0  
The rounding of 0.35 is 0  
The rounding of 0.3999999999999997 is 0  
The rounding of 0.4499999999999996 is 0  
The rounding of 0.4999999999999994 is 0  
The rounding of 0.5499999999999999 is 1  
The rounding of 0.6 is 1  
The rounding of 0.65 is 1  
The rounding of 0.7000000000000001 is 1  
The rounding of 0.7500000000000001 is 1  
The rounding of 0.8000000000000002 is 1  
The rounding of 0.8500000000000002 is 1  
The rounding of 0.9000000000000002 is 1  
The rounding of 0.9500000000000003 is 1
```

Formatting Numbers

- For formatting numbers, JAVA use a NumberFormat subclass.
- NumberFormat class is present in java.text package (i.e., import java.text.NumberFormat) and it is an abstract class.
- A DecimalFormat object appropriate to the user's locale can be obtained from the factory method NumberFormat.getInstance() and manipulated using set methods.
- *Creating a NumberFormat object*
 - NumberFormat nf = NumberFormat.getInstance()
- *Some important Set Methods of NumberFormat class*
 - void setMaximumFractionDigits(int newValue)

- Sets the maximum number of digits allowed in the fraction portion of a number.
- void setMaximumIntegerDigits(int newValue)
 - Sets the maximum number of digits allowed in the integer portion of a number.
- void setMinimumFractionDigits(int newValue)
 - Sets the minimum number of digits allowed in the fraction portion of a number.
- void setMinimumIntegerDigits(int newValue)
 - Sets the minimum number of digits allowed in the integer portion of a number.

Example: Write a program to print a double number having maximum fraction digit 2.

```
import java.text.NumberFormat;

public class NumberFormatTest {

    public static void main(String[] args) {

        NumberFormat nf=NumberFormat.getInstance();
        nf.setMaximumFractionDigits(2);

        double d=123.456;
        //The double number having maximum fraction digit 2.
        System.out.println("After formatting the number " + d + " is "
                           + nf.format(d));
    }
}
```

Output:

After formatting the number 123.456 is 123.46

Example: Write a program to print a double number having maximum fraction digit 4 and minimum fraction digit 2.

```
import java.text.NumberFormat;

public class NumberFormatTest {
```

```

public static void main(String[] args) {

    NumberFormat nf = NumberFormat.getInstance();
    nf.setMaximumFractionDigits(4);
    nf.setMinimumFractionDigits(2);

    double d=123.4;
    System.out.println("After formatting the number " + d + " is "
                       + nf.format(d));
    double e=12.14567;
    System.out.println("After formatting the number " + e + " is "
                       + nf.format(e));
}
}

```

Output:

After formatting the number 123.4 is 123.40
 After formatting the number 12.14567 is 12.1457

Example: Write a program to set minimum integer digit to 3, maximum fraction digit to 4 and minimum fraction digit to 2 of a decimal number.

```

import java.text.NumberFormat;

public class NumberFormatTest {

    public static void main(String[] args) {

        NumberFormat nf=NumberFormat.getInstance();
        nf.setMinimumIntegerDigits(3);
        nf.setMaximumFractionDigits(4);
        nf.setMinimumFractionDigits(2);

        double a=12.4;
        double b=121.14567;

        System.out.println("After formatting the number " + a + " is "
                           + nf.format(a));
        System.out.println("After formatting the number " + b + " is "
                           + nf.format(b));
    }
}

```

Output:

After formatting the number 12.4 is 012.40

After formatting the number 121.14567 is 121.1457

Example:

```

import java.text.NumberFormat;

public class NumberFormatTest {

    public static void main(String[] args) {

        final double data[] = {0, 1, 22d/7, 100.2345678};

        NumberFormat nf=NumberFormat.getInstance();

        nf.setMinimumIntegerDigits(3);
        nf.setMinimumFractionDigits(2);
        nf.setMaximumFractionDigits(4);

        // Now print using it.
        for (int i=0; i<data.length; i++) {
            System.out.println(data[i] + "formats as " + nf.format(data[i]));
        }
    }
}

```

Output:

0.0 formats as 000.00

1.0 formats as 001.00

3.142857142857143 formats as 003.1429

100.2345678 formats as 100.2346

Changing the pattern dynamically

You can also construct a DecimalFormat with a particular pattern or change the pattern dynamically using applyPattern() .

Table 5-2. DecimalFormat pattern characters

Character	Meaning
#	A digit, leading zeroes are omitted.
0	A digit - always displayed, even if number has less digits (then 0 is

	displayed)
.	Locale-specific decimal separator (decimal point)
,	Locale-specific grouping separator (comma in English)
-	Locale-specific negative indicator (minus sign)
%	Shows the value as a percentage
;	Separates two formats: the first for positive and the second for negative values
'	Escapes one of the above characters so it appears
Anything else	Appears as itself

Example: Write a program to print a double number in the format, 3 digit in hole part and decimal number then two digit in fraction part.

```

import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

    public static void main(String[] args) {

        NumberFormat our = new DecimalFormat("###.##");
        double d=123.345;
        System.out.println(d + "\tformats as " + our.format(d));
    }
}

```

Output:

123.34

check where,

Input: 12.456	Output: 12.46
Input: 12345.47789	Output: 12345.48
Input: .345	Output: 0.34

Example: Write a program to print a double number in the format, 4 digit before decimal(if number of digit is less print zero) and 3 digit after decimal.

```

import java.text.NumberFormat;
import java.text.DecimalFormat;

```

```

public class NumberFormatTest {

    public static void main(String[] args) {

        NumberFormat our = new DecimalFormat("0000.###");
        //NumberFormat our = new DecimalFormat("0000.000");
        double d=1234.5678;
        double e=12.5678;
        double f=12.5;
        //double g=7.5;
        System.out.println(d + "\tformats as " + our.format(d));
        System.out.println(e + "\tformats as " + our.format(e));
        System.out.println(f + "\tformats as " + our.format(f));
        //System.out.println(g + "\tformats as " + our.format(g));
                                //Output: 0007.500
    }
}

```

Output:

1234.5678 formats as 1234.568
 12.5678 formats as 0012.568
 12.5 formats as 0012.5

Example: Write a program to print a decimal number grouping it to 2 digit at a time.

```

import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

    public static void main(String[] args) {
        NumberFormat our = new DecimalFormat("##,##.##");
        double d=1245677.5566;
        System.out.println(d + "\tformats as " + our.format(d));
    }
}

```

Output:

1245677.5566 formats as 1,24,56,77.56

Example: Write a program to print percentage of a double number.

```
import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

    public static void main(String[] args) {

        NumberFormat our = new DecimalFormat("%##.##");
        //Write a program to print a double number -ve sign
        //assigned to it.
        //NumberFormat our = new DecimalFormat("-##.##");
        double d=657.5566;
        System.out.println(our.format(d));
    }
}
```

Output:

```
%65755.66
```

Converting Between Binary, Octal, Decimal, and Hexadecimal

- **Integer.parseInt(String input, int radix)** to convert from any type of number to an Integer object.
- **Integer.valueOf(String input, int radix)** is an inbuilt method which convert any type of number to returns an Integer object.
- **Integer.toString(int input, int radix)** to convert from integer to any type. The return type is string.
 - There are also specialized versions of `toString(int)` that don't require you to specify the radix; for example, **toBinaryString()** to convert an integer to binary, **toHexString()** for hexadecimal, and so on.

Example: Convert the string "1010" to decimal number. here the radix are binary, octal, hexadecimal. Hint: by using `Integer.parseInt(String input, int radix)`

```

import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

    public static void main(String[] args) {

        String str="1010";
        Integer iObj=Integer.parseInt(str,2);
        System.out.println("1010 in base 2 is "+iObj);
        Integer iObj1=Integer.parseInt(str,8);
        System.out.println("1010 in base 8 is "+iObj1);
        Integer iObj2=Integer.parseInt(str,16);
        System.out.println("1010 in base 16 is "+iObj2);
    }
}

```

Output:

1010 in base 2 is 10
 1010 in base 8 is 520
 1010 in base 16 is 4112

Example: Convert the number 42 to the binary, octal, hexadecimal and decimal number using Integer.toString(int input, int radix).

```

import java.text.NumberFormat;
import java.text.DecimalFormat;

public class NumberFormatTest {

    public static void main(String[] args) {
        int i=42;
        String res1=Integer.toString(i,2);
        String res2=Integer.toString(i,8);
        String res3=Integer.toString(i,16);
        String res4=Integer.toString(i,10);
        System.out.println("42 in base 2 is "+res1);
        System.out.println("42 in base 8 is "+res2);
        System.out.println("42 in base 16 is "+res3);
        System.out.println("42 in base 10 is "+res4);

        String res5=Integer.toBinaryString(i);
        System.out.println("42 in base 2 is "+res5);
    }
}

```

```
    }  
}
```

Output:

```
42 in base 2 is 101010  
42 in base 8 is 52  
42 in base 16 is 2a  
42 in base 10 is 42  
42 in base 2 is 101010
```

Example:

```
import java.text.NumberFormat;  
import java.text.DecimalFormat;  
  
public class NumberFormatTest {  
  
    public static void main(String[] args) {  
        String input = "101010";  
        for (int radix : new int[] { 2, 8, 10, 16, 36 }) {  
            System.out.print(input + " in base " + radix + " is " +  
                Integer.valueOf(input, radix) + "; ");  
  
            Int j = 42;  
            System.out.println(j + " formatted in base " + radix + " is "  
                + Integer.toString(j, radix));  
        }  
    }  
}
```

Output:

```
101010 in base 2 is 42;           42 formatted in base 2 is 101010  
101010 in base 8 is 33288;       42 formatted in base 8 is 52  
101010 in base 10 is 101010;     42 formatted in base 10 is 42  
101010 in base 16 is 1052688;    42 formatted in base 16 is 2a  
101010 in base 36 is 60512868;   42 formatted in base 36 is 16
```

Operating on a Series of Integers

- Need to work on a range of integers.
 - If the set of numbers are contiguous, use a for loop.

- If the ranges of numbers are discontinuous, use a `java.util.BitSet` class.

Example: For a contiguous set, use for loop.

```

public class Number {

    protected static String months[] = {"January", "February", "March",
    "April", "May", "June", "July", "August", "September", "October",
    "November", "December"};

    public static void main(String[] args) {

        // When you want a set of array indices, use a for loop starting at 0.
        for (int i = 0; i < months.length; i++)
            System.out.println("Month " + months[i]);

        System.out.println(" ");

        // When you want an ordinal list of numbers, use a for loop starting at 1.
        for (int i = 1; i <= months.length; i++)
            System.out.println("Month # " + i);

        System.out.println(" ");

        // For e.g., counting by 3 from 11 to 27, use a for loop
        for (int i = 11; i <= 27; i += 3)
            System.out.println("i = " + i);
    }
}

```

Output:

```

Month January
Month February
Month March
Month April
Month May
Month June
Month July
Month August
Month September
Month October
Month November
Month December

```

```
Month # 1
Month # 2
Month # 3
Month # 4
Month # 5
Month # 6
Month # 7
Month # 8
Month # 9
Month # 10
Month # 11
Month # 12
```

```
i = 11
i = 14
i = 17
i = 20
i = 23
i = 26
```

Example: For discontinuous ranges of numbers, use a `java.util.BitSet` class.

- **BitSet** is a class defined in the `java.util` package. It creates an array of bits represented by boolean values.
- The size of the array is flexible and can grow to accommodate additional bit as needed.
- As it is an array, the index is zero-based and the bit values can be accessed only by non-negative integers as an index .
- The default value of the BitSet is boolean false with a representation as 0 (off).
- **BitSet()** : A no-argument constructor to create an empty BitSet object.
- BitSet uses about 1 bit per boolean value.
- **set(int Index)** : This method sets the bit at the specified index to true i.e adds a value.

- To access a specific value in the BitSet, the **get(int index)** method is used with an integer argument as an index.

```
import java.util.BitSet;

public class Number {

    protected static String months[] = {"January", "February", "March",
    "April", "May", "June", "July", "August", "September", "October",
    "November", "December"};

    public static void main(String[] args) {

        // Create a BitSet and turn on a couple of bits.
        BitSet b = new BitSet();
        b.set(0); // January
        b.set(3); // April
        b.set(8); // September

        // Presumably this would be somewhere else in the code.
        for (int i = 0; i < months.length; i++) {
            if (b.get(i))
                System.out.println("Month " + months[i]);
        }

        System.out.println();

        // Same example using an array
        int[] numbers = {0, 3, 8};

        // Presumably this would be somewhere else in the code.
        for (int n : numbers) {
            System.out.println("Month: " + months[n]);
        }
    }
}
```

Output:

Month January
 Month April
 Month September

Month: January

Month: April

Month: September

Formatting with Correct Plurals

- The examples of use of correct plurals are
 - 0 Books or Zero Books
 - 1 Book or One Book
 - 2 Books or Two Books
- We can solve the problem by using one of the following method
 - Using **conditional statement**, or
 - Using **ChoiceFormat**

Using conditional statement

Example: Using conditional statement

```
public class Number {
```

```
    public static void main(String[] args) {  
  
        //int n=0;  
        int n=1;  
        //int n=2;  
        System.out.println("I read " + n + " Book" + (n==1? "." : "s."));  
    }  
}
```

Output:

I read 1 Book.

Using ChoiceFormat

- Using *ChoiceFormat with pluralizedFormat*
 - ChoiceFormat accepts two arrays

- It provides pluralized word
- Using *ChoiceFormat with quantizedFormat*
 - ChoiceFormat accepts string-based pattern
 - English text version of quantity

Example: Using ChoiceFormat with pluralizedFormat

```
import java.text.ChoiceFormat;

public class Number {

    // ChoiceFormat to just give pluralized word
    static double[] limits = { 0, 1, 2 };

    static String[] formats = { "reviews", "review", "reviews" };

    static ChoiceFormat pluralizedFormat = new
        ChoiceFormat(limits, formats);

    Static int[] data = { -2,-1, 0, 1, 2, 3};

    public static void main(String[] args) {

        System.out.println("Pluralized Format");

        for (int i : data) {
            System.out.println("Found " + i + " " +
                pluralizedFormat.format(i));
        }
    }
}
```

Output:

```
Pluralized Format
Found -2 reviews
Found -1 reviews
Found 0 reviews
Found 1 review
Found 2 reviews
Found 3 reviews
```

Example: Using ChoiceFormat with quantizedFormat

```
import java.text.ChoiceFormat;

public class Number {

    // ChoiceFormat to give English text version, quantified
    static ChoiceFormat quantizedFormat = new ChoiceFormat(
        "0#no reviews | 1#one review | 1<many reviews");

    static int[] data = { -2, -1, 0, 1, 2, 3 };

    public static void main(String[] args) {

        System.out.println("Quantized Format");

        for (int i : data) {
            System.out.println("Found " +
                quantizedFormat.format(i));
        }
    }
}
```

Output:

```
Quantized Format
Found no reviews
Found no reviews
Found no reviews
Found one review
Found many reviews
Found many reviews
```

Generating Random Numbers

Java provides 2 techniques to generate random numbers using some built-in methods and classes as listed below:

- *Math.random method*
- *java.util.Random class*

Math.random method :

- This technique can generate random numbers of double type.
- It is a positive sign, greater than or equal to 0.0 and less than 1.0.
- Use java.lang.Math.random()

Example:

```
public class Number {  
  
    public static void main(String[] args) {  
  
        System.out.println("A random from java.lang.Math is " +  
                           Math.random( ));  
        System.out.println("A random from java.lang.Math is " +  
                           Math.random( ));  
    }  
}
```

Output:

A random from java.lang.Math is 0.5329005395702993
A random from java.lang.Math is 0.7750836085959537

java.util.Random

- We can generate random numbers of types integers, float, double, long, booleans using this class.
- To generate random numbers
 - first create an instance of this class
 - then invoke methods such as nextInt(), nextDouble(), nextLong() etc using that instance.
- We can pass arguments to the methods for placing an upper bound on the range of the numbers to be generated.
 - For example, nextInt(6) will generate numbers in the range 0 to 5 both inclusive.

Example:

```
import java.util.Random;

public class Number {

    public static void main(String[] args) {

        Random rand = new Random();

        // Generate random integers in range 0 to 999
        int rand_int1 = rand.nextInt(1000);
        int rand_int2 = rand.nextInt(1000);

        // Print random integers
        System.out.println("Random Integers: "+rand_int1);
        System.out.println("Random Integers: "+rand_int2);

        // Generate Random doubles
        double rand_dub1 = rand.nextDouble();
        double rand_dub2 = rand.nextDouble();

        // Print random doubles
        System.out.println("Random Doubles: "+rand_dub1);
        System.out.println("Random Doubles: "+rand_dub2);

        boolean rand_bool1 = rand.nextBoolean();
        System.out.println("Random Booleans: "+ rand_bool1);

    }
}
```

Output:

```
Random Integers: 730
Random Integers: 751
Random Doubles: 0.7653875428555855
Random Doubles: 0.013700301907223489
Random Booleans: false
```

Example: write a program to generate 3 double type random numbers and 3 integer type random numbers.

Output:

```
A gaussian random double is -0.12654554740642937  
A gaussian random double is -1.3442928221637291  
A gaussian random double is -0.2011491511058577  
A gaussian random double is 0.023075071226920853  
A gaussian random double is 1.0032986463793938
```

Note: A Gaussian or normal distribution is a bell- curve of values from negative infinity to positive infinity, with the majority of the values around zero (0.0).

Calculating Trigonometric Functions

- Use `java.lang.Math` class.
- For compute sine, cosine, and other trigonometric functions.
- Note that the arguments for trigonometric functions are in radians, not in degrees.
- $\text{Degree} = \text{radian} * (180/\text{PI})$

Example:

```
import java.util.Random;  
  
public class Number {  
  
    public static void main(String[] args) {  
  
        System.out.println("The cosine of 1.1418 is " + Math.cos(1.1418));  
  
        System.out.println("The sine of PI/4 is " + Math.sin(Math.PI/4));  
  
        System.out.println("Java's PI is " + Math.PI);  
  
        System.out.println("Java's e is " + Math.E);  
  
    }  
}
```

Output:

```
The cosine of 1.1418 is 0.41595828804562746
```

The sine of 1.1418 is 0.7071067811865475
Java's PI is 3.141592653589793
Java's e is 2.718281828459045

Taking Logarithms

- Find the logarithm of a number. It can be 2 types:
 - *logarithms to base e*
 - *logarithms to other bases*

logarithms to base e

- The `java.lang.Math.log(double a)` returns the natural logarithm (base e) of a double value.
- Use `java.lang.Math.log()` method and the syntax is
 - *public static double log(double a)*
 - *here a is a value*
- *Special cases:*
 - If the argument is NaN or less than zero, then the result is NaN.
 - If the argument is positive infinity, then the result is positive infinity.
 - If the argument is positive zero or negative zero, then the result is negative infinity.

Example:

```
public class Number {  
  
    public static void main(String[] args) {  
  
        // get two double numbers  
        double x = 60984.1;  
        double y = -497.99;
```

```

// get the natural logarithm for x
System.out.println("Math.log(" + x + ")=" + Math.log(x));

// get the natural logarithm for y
System.out.println("Math.log(" + y + ")=" + Math.log(y));
}
}

```

Output:

Math.log(60984.1)=11.018368453441132
 Math.log(-497.99)=NaN

logarithms to other bases

- The logarithms of other base can be determined by the following equation
- $$\log_n (x) = \frac{\log_e (x)}{\log_e (n)}$$
- where x is the number whose logarithm you want, n is any desired base, and e is the natural logarithm base.

Example: Write the program to find $\log_{10}(10000)$

```

public class Number {

    public static double log_base(double base, double value) {

        return Math.log(value) / Math.log(base);
    }

    public static void main(String[] args) {

        double d = Number.log_base(10, 10000);

        System.out.println("log10(10000) = " + d);

    }
}

```

Output:

$\log_{10}(10000) = 4.0$

Multiplying Matrices

Example: Write a program to multiply two matrices.

```
public class Number {  
  
    public static int[][] multiply(int[][] m1, int[][] m2) {  
  
        int m1rows = m1.length;  
  
        int m1cols = m1[0].length;  
  
        int m2rows = m2.length;  
  
        int m2cols = m2[0].length;  
  
        if (m1cols != m2rows)  
            throw new IllegalArgumentException("matrices  
                don't match: " + m1cols + " != " + m2rows);  
  
        int[][] result = new int[m1rows][m2cols];  
  
        // multiply  
        for (int i=0; i<m1rows; i++) {  
            for (int j=0; j<m2cols; j++) {  
                for (int k=0; k<m1cols; k++) {  
  
                    result[i][j] += m1[i][k] * m2[k][j];  
                }  
            }  
        }  
        return result;  
    }  
  
    /** Matrix print. */  
    public static void mprint(int[][] a) {  
  
        int rows = a.length;  
  
        int cols = a[0].length;  
  
        System.out.println("array["+rows+"]["+cols+"] = {");  
    }  
}
```

```

for (int i=0; i<rows; i++) {
    System.out.print("{");
    for (int j=0; j<cols; j++) {
        System.out.print(" " + a[i][j] + ",");
    }
    System.out.println("},");
}
System.out.println("};");
}

public static void main(String[] args) {
    int x[][] = {
        { 3, 2, 3 },
        { 5, 9, 8 },
    };
    int y[][] = {
        { 4, 7 },
        { 9, 3 },
        { 8, 1 },
    };
    int z[][] = Number.multiply(x, y);
    Number.mprint(x);
    Number.mprint(y);
    Number.mprint(z);
}
}

```

Output:

```

{ 3, 2, 3 },
{ 5, 9, 8 },
};
array[3][2] = {
{ 4, 7 },

```

```

{ 9, 3, },
{ 8, 1, },
};
array[2][2] = {
{ 54, 30, },
{ 165, 70, },
};

```

Handling Very Large Numbers

In java two classes to handle the large number

- BigInteger
 - It creates a large integer number and handles integer numbers larger than Long.MAX_VALUE.
- BigDecimal
 - It creates a large decimal number (Real number) and handle floating-point values larger than Double.MAX_VALUE.
- Use the BigInteger or BigDecimal values in java.math package.

BigInteger

- It creates a large integer number and handles integer numbers larger than Long.MAX_VALUE.
- Need to import java.math.BigInteger
- Create BigInteger instance as follows:
 - BigInteger bi=new BigInteger(String val);
- It consists of many methods, but some of the methods are

Methods	Descriptions
<u>abs()</u>	It returns a BigInteger whose value is the absolute value of this BigInteger.
<u>add()</u>	This method returns a BigInteger by simply computing 'this + val' value.

<u>subtract()</u>	This method returns a BigInteger by simply computing 'this - val' value.
<u>multiply()</u>	This method returns a BigInteger by computing 'this *val' value.
<u>divide()</u>	This method returns a BigInteger by computing 'this /~val' value.
<u>pow()</u>	This method returns a BigInteger whose value is 'this ^{exponent} '.
<u>negate()</u>	This method returns a BigInteger whose value is '-this'.
<u>compareTo()</u>	Compares this BigInteger with the specified BigInteger.
<u>equals()</u>	Compares this BigInteger with the specified Object for equality.
<u>floatValue()</u>	Converts this BigInteger to a float.
<u>intValue()</u>	Converts this BigInteger to a int.

Example:

```

import java.math.BigInteger;

public class VeryLargeNumber {

    public static void main(String[] args) {

        // Create two new BigInteger
        BigInteger BI1 = new BigInteger("123445566645676532");
        BigInteger BI2 = new BigInteger("98765432187543678289");
        BigInteger BI3 = new BigInteger("-532987654321123456789");
        BigInteger BI20 = new BigInteger("123445");

        // Absolute value of BigInteger
        BigInteger BI4 = BI3.abs();
        System.out.println("BigInteger Absolute value result = " + BI4);

        // Addition of two BigIntegers
        BigInteger BI5 = BI1.add(BI2);
        System.out.println("BigInteger Addition result = " + BI5);

        // Multiplication of two BigIntegers
        BigInteger BI6 = BI1.multiply(BI2);
        System.out.println("BigInteger Multiplication result= " + BI6);
    }
}

```

```

// Subtraction of two BigIntegers
BigInteger BI7 = BI1.subtract(BI2);
System.out.println("BigInteger Subtract result = " + BI7);

// Division of two BigIntegers
BigInteger BI8 = BI2.divide(bd2);
System.out.println("BigInteger Division result = " + BI8);

// BigInteger raised to the power of 2
BigInteger BI9= BI1.pow(2);
System.out.println("BigInteger Exponent result = " + BI9);

// Negate value of BigInteger1
BigInteger BI10 = BI1.negate();
System.out.println("BigInteger Negation result = " + BI10);

// Compare the value of BigIntegers
int BI11 = BI1.compareTo(BI2);
System.out.println("BigInteger Compare result = " + BI 11);
                                // 0 or 1 or -1

// equals value of BigInteger
Boolean bd12 = bd1.equals(bd2);
System.out.println("BigInteger Equal result = " + bd12);
                                // false or true

int BI13 = BI20.intValue();
System.out.println("Integer result = " + BI13);

float BI14 = BI1.floatValue();
System.out.println("Float result = " + BI14);
}

}

```

Output:

```

BigInteger Absolute value result = 532987654321123456789
BigInteger Addition result = 98888877754189354821
BigInteger Multiplication result=
12192154741396469252441892370165213748
BigInteger Subtract result = -98641986620898001757

```

```

BigInteger Division result = 1
BigInteger Exponent result = 15238807924472166308212407975547024
BigInteger Negation result = -123445566645676532
BigInteger Compare result = -1
BigInteger Equal result = false
Integer result = 123445
Float result = 1.23445563E17

```

BigDecimal

- It creates a large decimal number (Real number) and handle floating-point values larger than Double.MAX_VALUE.
- Need to import java.math.BigDecimal
- Create BigInteger instance as follows:
 - BigDecimal bi=new BigDecimal(String val)
- It consists of many methods, but some of the methods are

Methods	Descriptions
<u>abs()</u>	It returns a BigDecimal whose value is the absolute value of this BigDecimal.
<u>add()</u>	This method returns a BigDecimal by simply computing 'this + val' value.
<u>subtract()</u>	This method returns a BigDecimal by simply computing 'this - val' value.
<u>multiply()</u>	This method returns a BigDecimal by computing 'this *val' value.
<u>divide()</u>	This method returns a BigDecimal by computing 'this /~val' value.
<u>pow()</u>	This method returns a BigDecimal whose value is 'this ^{exponent} '.
<u>negate()</u>	This method returns a BigDecimal whose value is '- this'.
<u>compareTo()</u>	Compares this BigDecimal with the specified BigDecimal.
<u>equals()</u>	Compares this BigDecimal with the specified Object for equality.

Example:

```
import java.math.BigDecimal;

public class VeryLargeNumber {

    public static void main(String[] args) {

        // Create two new BigDecimals
        BigDecimal BD1 = new BigDecimal("124567890.0987654321");
        BigDecimal BD2 = new BigDecimal("987654321.123456789");
        BigDecimal BD3 = new BigDecimal("-532987654321.123456");

        // Absolute value of BigDecimal
        BigDecimal BD4 = BD3.abs();
        System.out.println("BigDecimal Absolute value result = " + BD4);

        // Addition of two BigDecimals
        BigDecimal BD5 = BD1.add(BD2);
        System.out.println("BigDecimal Addition result = " + BD5);

        // Multiplication of two BigDecimals
        BigDecimal BD6 = BD1.multiply(BD2);
        System.out.println("BigDecimal Multiplication result= " + BD6);

        // Subtraction of two BigDecimals
        BigDecimal BD7 = BD1.subtract(BD2);
        System.out.println("BigDecimal Subtract result = " + BD7);

        // Division of two BigDecimals
        BigDecimal BD8 = BD2.divide(BD2);
        System.out.println("BigDecimal Division result = " + BD8);

        // BigDecimal1 raised to the power of 2
        BigDecimal BD9= BD1.pow(2);
        System.out.println("BigDecimal Exponent result = " + BD9);

        // Negate value of BigDecimal1
        BigDecimal BD10 = BD1.negate();
        System.out.println("BigDecimal Negation result = " + BD10);

        // Negate value of BigDecimal1
        int BD11 = BD1.compareTo(BD2);
```

```

        System.out.println("BigDecimal Compare result = " + BD11);
                                            // 0 or 1 or -1

        // Negate value of BigDecimal1
        Boolean BD12 = BD1.equals(BD2);
        System.out.println("BigDecimal Equal result = " + BD12);
                                            // false or true
    }

}

```

Output:

BigDecimal Absolute value result = 532987654321.123456789
 BigDecimal Addition result = 1112222211.2222222211
 BigDecimal Multiplication result=
 123030014929277547.5030955772112635269
 BigDecimal Subtract result = -863086431.0246913569
 BigDecimal Division result = 1
 BigDecimal Exponent result = 15517159243658102.97302514857789971041
 BigDecimal Negation result = -124567890.0987654321
 BigDecimal Compare result = -1
 BigDecimal Equal result = false

Program: TempConverter

The following program prints a table of Fahrenheit temperatures and the corresponding Celsius temperatures by using *printf*, which control the formatting of the converted temperatures.

Example:

```

public class TempConverter {

    public static void main(String[] args) {
        TempConverter t = newTempConverter();
        t.start();
        t.data();
        t.end();

    }
}

```

```

protected void start() {
    System.out.println(" Fahr Centigrade");
}

protected void data() {
    for (inti=-40; i<=120; i+=10) {
        floatc = (i-32)*(5f/9);
        print(i, c);
    }
}

protected void print(floatf, floatc) {
    System.out.printf("%6.2f %6.2f%n", f, c);
}

protected void end() {
    System.out.println("-----");
}

```

Output:

Fahr Centigrade

-40.00	-40.00
-30.00	-34.44
-20.00	-28.89
-10.00	-23.33
0.00	-17.78
10.00	-12.22
20.00	-6.67
30.00	-1.11
40.00	4.44
50.00	10.00
60.00	15.56
70.00	21.11
80.00	26.67
90.00	32.22
100.00	37.78
110.00	43.33
120.00	48.89

Exception Handling

By Dr. Subrat Kumar Nayak
Associate Professor
Department of CSE
ITER, SOADU

Exception Handling in Java

- The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

What is Exception in Java?

- Exception is an abnormal condition.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

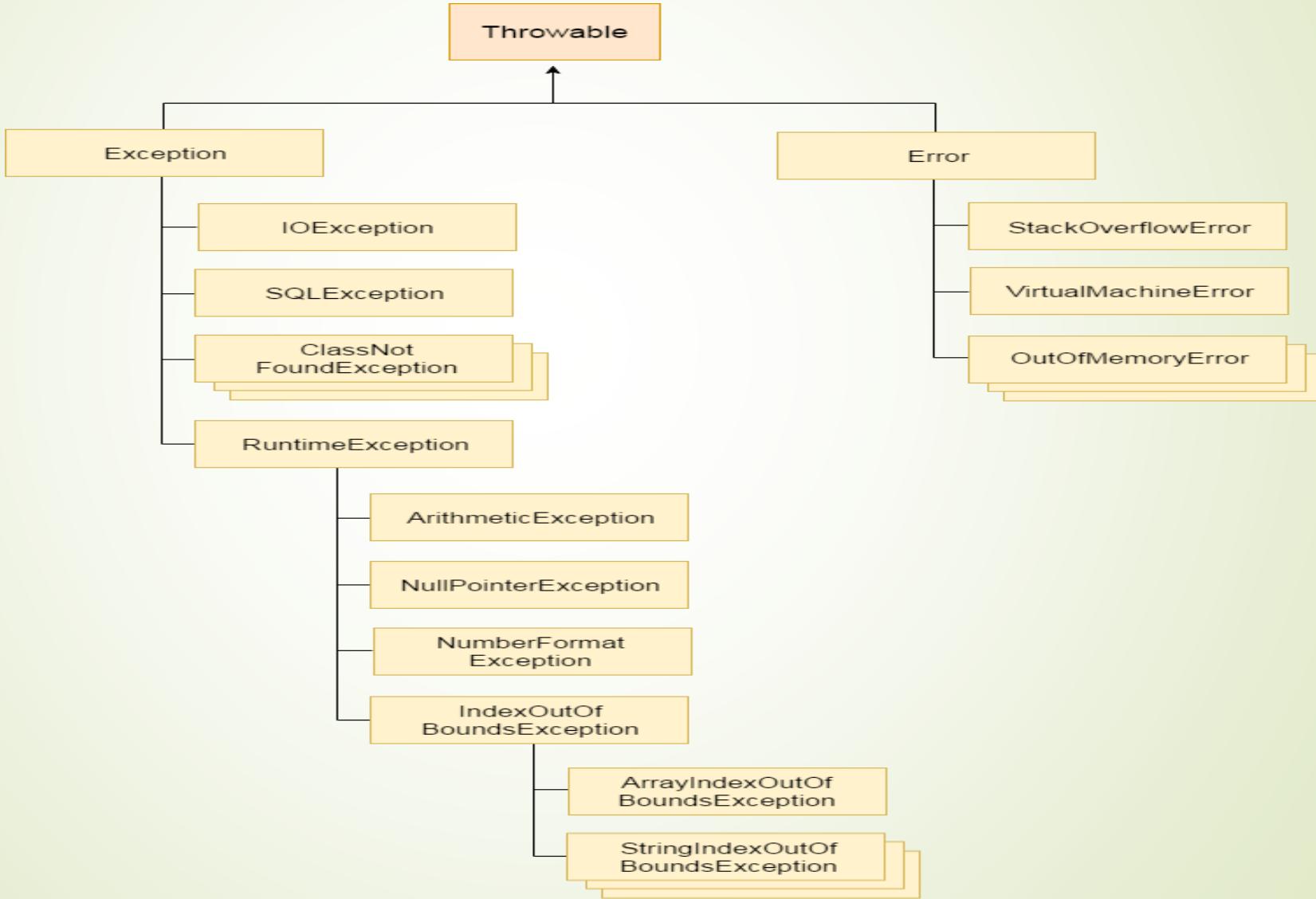
What is Exception Handling?

- Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Advantage

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling.

Hierarchy of Java Exception classes



Types of Java Exceptions

- There are mainly two types of exceptions: checked and unchecked. Here, an error is considered as the unchecked exception. According to Oracle, there are three types of exceptions:
 1. Checked Exception
 2. Unchecked Exception
 3. Error

Checked Exception

- The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions e.g. IOException, SQLException etc.
- Checked exceptions are checked at **compile-time**.

Unchecked Exception

- The classes which inherit RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
- Unchecked exceptions are not checked at compile-time, but they are checked at **runtime**.

Error

- Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

Java Exception Keywords

try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

Exception Handling in Java

Example: (Java Exception Handling using a try-catch statement)

```
public class JavaExceptionExample{  
    public static void main(String args[]){  
        try{  
            //code that may raise exception  
            int data=100/0;  
        }catch(ArithmetricException e){System.out.println(e);}  
        //rest code of the program  
        System.out.println("rest of the code...");  
    }  
}
```

Common Scenarios of Java Exceptions

- ▶ A scenario where ArithmeticException occurs

```
int a=50/0;//ArithmeticException
```

- ▶ A scenario where NullPointerException occurs

```
String s=null;
```

```
System.out.println(s.length());//NullPointerException
```

- ▶ A scenario where NumberFormatException occurs

```
String s="abc";
```

```
int i=Integer.parseInt(s);//NumberFormatException
```

- ▶ A scenario where ArrayIndexOutOfBoundsException occurs

```
int a[]={new int[5];
```

```
a[10]=50; //ArrayIndexOutOfBoundsException
```

Exception Handling in Java

Java try-catch block

try block

- ▶ Java **try** block is used to enclose the code that might throw an exception.
- ▶ If an exception occurs at the particular statement of try block, the rest of the block code will not execute.
- ▶ Java try block must be followed by either catch or finally block.

Syntax:

- ▶ try-catch

```
try{
```

```
    //code that may throw an exception
```

```
}catch(Exception_class_Name ref){}
```

- ▶ try-finally

```
try{
```

```
    //code that may throw an exception
```

```
}finally{}
```

Exception Handling in Java

catch block

- ▶ Java catch block is used to handle the Exception by declaring the type of exception within the parameter.
- ▶ The declared exception must be the parent class exception (i.e., **Exception**) or the **generated exception** type.
- ▶ The catch block must be used after the try block only. You can use **multiple catch block** with a single try block.

Problem without exception handling

```
public class TryCatchExample1 {  
    public static void main(String[] args) {  
        int data=50/0; //may throw exception  
        System.out.println("rest of the code");  
    } }
```

Output: Exception in thread "main" java.lang.ArithmetricException: / by zero

Exception Handling in Java

Solution by exception handling

```
public class TryCatchExample2 {  
    public static void main(String[] args) {  
        try  
        {  
            int data=50/0; //may throw exception  
        }  
        catch(ArithmaticException e)  
        {  
            System.out.println(e); / System.out.println("Can't divide by zero");  
        }  
        System.out.println("rest of the code");  
    } }  
Output: java.lang.ArithmaticException: / by zero Or Can't divide by zero  
rest of the code rest of the code
```

Exception Handling in Java

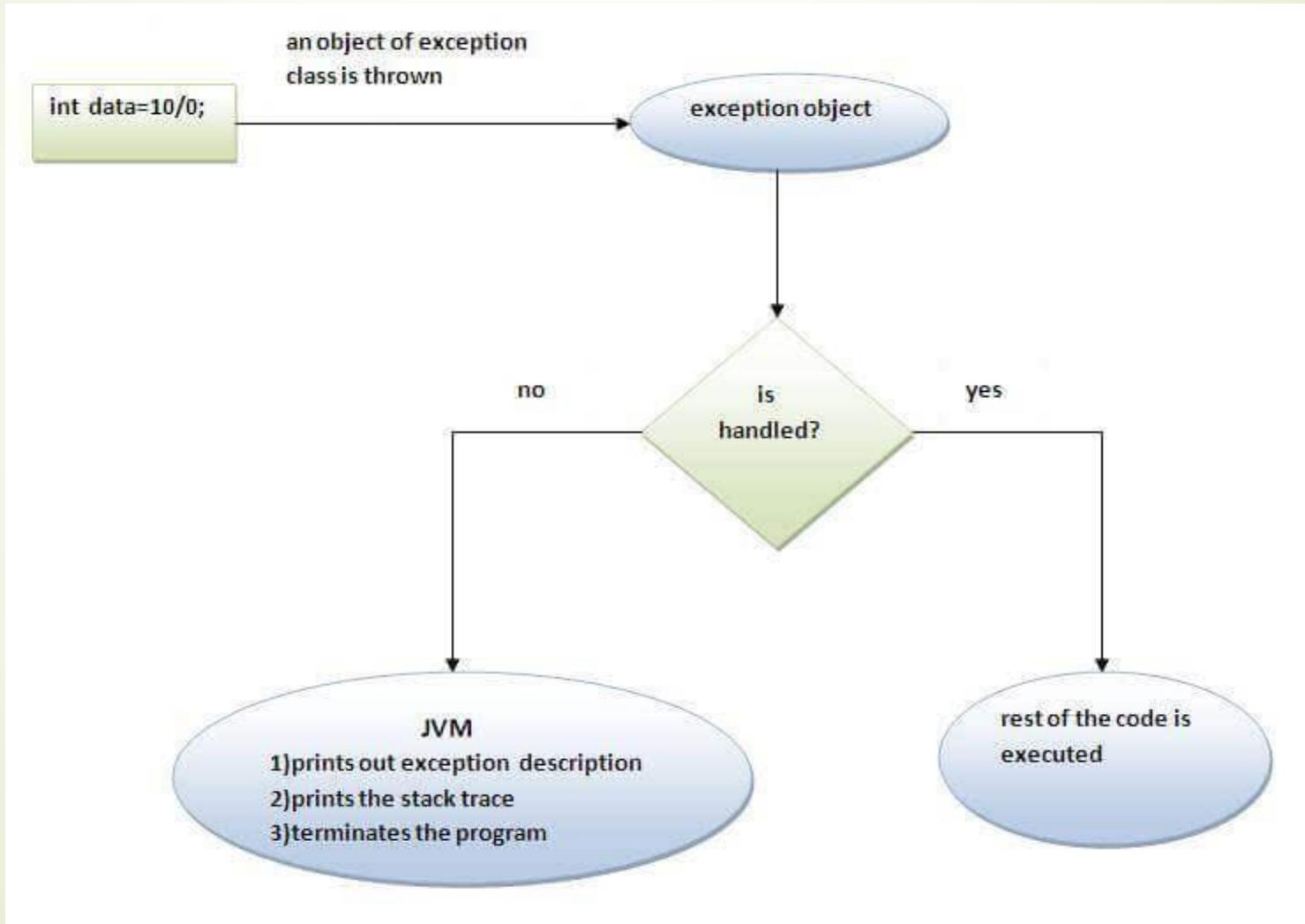
Example: Handling with different type of exception class

```
public class TryCatchExample8 {  
    public static void main(String[] args) {  
        try {  
            int data=50/0; //may throw exception  
        }  
        catch(ArrayIndexOutOfBoundsException e)  
        {  
            System.out.println(e);  
        }  
        System.out.println("rest of the code");  
    } }
```

Output: Exception in thread "main" java.lang.ArithmetricException: / by zero

Exception Handling in Java

Internal working of java try-catch block



Exception Handling in Java

Java Multi-catch block

- ▶ A try block can be followed by one or more catch blocks. Each catch block must contain a different exception handler. So, if you have to perform different tasks at the occurrence of different exceptions, use java multi-catch block.

```
public class MultipleCatchBlock1 {  
    public static void main(String[] args) {  
        try{  
            int a[]={new int[5];  
            a[5]=30/0;  
        }  
        catch(ArithmaticException e)  
        {  
            System.out.println("Arithmatic Exception occurs");  
        }  
    }  
}
```

```
catch(ArrayIndexOutOfBoundsException e)  
{  
    System.out.println("ArrayIndexOutOfBoundsException occurs");  
}  
catch(Exception e)  
{  
    System.out.println("Parent Exception occurs");  
}  
System.out.println("rest of the code");  
}
```

Exception Handling in Java

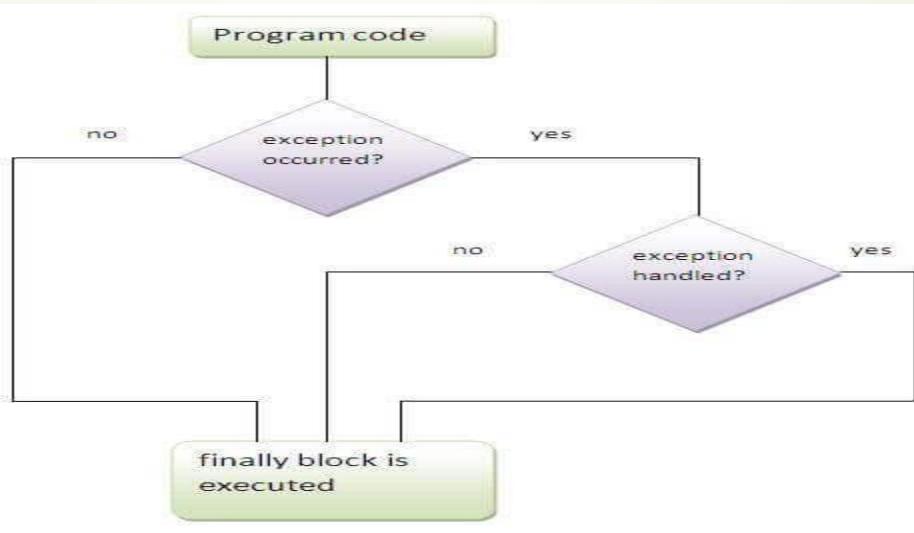
Java finally block

- ▶ **Java finally block** is a block that is used to execute important code such as closing connection, stream etc.
- ▶ Java finally block is always executed whether exception is handled or not.
- ▶ Java finally block follows try or catch block.

```
class TestFinallyBlock1{  
    public static void main(String args[]){  
        try{  
            int data=25/0;  
            System.out.println(data);  
        }  
        catch(NullPointerException e){System.out.println(e);}  
        finally{System.out.println("finally block is always executed");}  
        System.out.println("rest of the code..."); //will not execute  
    } }  
Output: finally block is always executed
```

Exception in thread main java.lang.ArithmetricException:/ by zero

Exception Handling in Java



- ▶ If you don't handle exception, before terminating the program, JVM executes finally block(if any).
- ▶ For each try block there can be zero or more catch blocks, but only one finally block.
- ▶ The finally block will not be executed if program exits(either by calling `System.exit()` or by causing a fatal error that causes the process to abort).

Why use java finally?

- ▶ Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

Exception Handling in Java

Java throw keyword

- ▶ The Java throw keyword is used to explicitly throw an exception.
- ▶ We can throw either checked or unchecked exception in java by throw keyword.
- ▶ The throw keyword is **mainly used to throw custom exception.**

Syntax

- ▶ **throw** exception;

Example

```
public class TestThrow1{
    static void validate(int age){
        if(age<18)
            throw new ArithmeticException("not valid");
        else
            System.out.println("welcome to vote");
    }
    public static void main(String args[]){
        validate(13);
        System.out.println("rest of the code...");
    }
}
```

Output: Exception in thread main java.lang.ArithmeticException:not valid

Exception Handling in Java

Java Custom Exception

- ▶ If you are creating your own Exception that is known as custom exception or user-defined exception.
- ▶ Java custom exceptions are used to customize the exception according to user need.
- ▶ By the help of custom exception, you can have your own exception and message.

Example:

```
class InvalidAgeException extends Exception{  
    InvalidAgeException(String s){  
        super(s);  
    }  
}
```

Output: Exception occurred: InvalidAgeException: not valid
rest of the code...

```
class TestCustomException1{  
  
    static void validate(int age) throws InvalidAgeException{  
        if(age<18)  
            throw new InvalidAgeException("not valid");  
        else  
            System.out.println("welcome to vote");  
    }  
  
    public static void main(String args[]){  
        try{  
            validate(13);  
        }catch(Exception m){System.out.println("Exception occurred: "+m);}  
  
        System.out.println("rest of the code...");  
    }  
}
```



End of Session

DATE and TIME

- Java has introduced a new Date and Time API since Java 8.
- To work with the date and time API, need to import the *java.time* package.
- The package includes many date and time classes.

Display Current Date

- To display the current date, import the *java.time.LocalDate* class, and use its *now()* method.

```
import java.time.LocalDate;

public class DateTime {

    public static void main(String[] args) {

        // Create a date object
        LocalDate ObjDate = LocalDate.now();

        // Display the current date in YYYY-MM-DD
        System.out.println("Today's Date: " + ObjDate);
    }
}
```

Output:

Today's Date: 2020-03-08

Display Current Time

- To display the current time (hour, minute, second, and milliseconds), import the *java.time.LocalTime* class, and use its *now()* method.

```
import java.time.LocalTime;

public class DateTime {

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

```

// Create a time object
LocalTime ObjTime = LocalTime.now();

// Display the current time in HH-MIN-SEC-MS
System.out.println("Current Time: " + ObjTime);//
}
}

```

Output:

Current Time: 20:46:49.375

Display Current Date and Time

- To display the current date and time, import the *java.time.LocalDateTime* class, and use its *now()* method.

```
import java.time.LocalDateTime;
```

```
public class DateTime {
```

```

public static void main(String[] args) {

    // Create a time object
    LocalDateTime ObjDateTime = LocalDateTime.now();

    // Display the current Date in YYYY-MM-DD and
    // time in HH-MIN-SEC-MS
    System.out.println("Current Date and Time: " + ObjDateTime);
}
```

Output:

Current Date and Time: 2020-03-08T20:52:30.005

Formatting Date and Time

- To provide better formatting for date and time objects
- Use *java.time.format.DateTimeFormatter* class with the *ofPattern()* method.

```

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class DateTime {

    public static void main(String[] args) {

        LocalDateTime myDateObj = LocalDateTime.now();
        System.out.println("Before formatting: " + myDateObj);

        DateTimeFormatter myFormatObj =
            DateTimeFormatter.ofPattern("dd-M-yyyy HH:mm:ss");

        String formattedDate = myDateObj.format(myFormatObj);
        System.out.println("After formatting: " + formattedDate);
    }
}

```

Output:

Before formatting: 2020-03-08T21:20:31.944

After formatting: 08-03-2020 21:20:31

<i>DateFormatter format characters</i>			
Symbol	Meaning	Presentation	Examples
G	Era	Text	AD; Anno Domini
y	Year-of-era	Year	2004; 04
u	Year-of-era	Year	y and u work the same for A.D. years; however, for a year of 3 B.C.,y pattern returns 3, whereas u pattern returns -2 (aka proleptic year).
D	Day-of-year	Number	189
M/L	Month-of-year	Number/text	7; 07; Jul; July; J
d	Day-of-month	Number	10
Q/q	Quarter-of-year	Number/text	3; 03; Q3, 3rd quarter
Y	Week-based-year	Year	1996; 96
w	Week-of-week-based year	Number	27
W	Week-of-month	Number	4
e/c	Localized day-of-week	Number/text	2; 02; Tue; Tuesday; T

E	Day-of-week	Text	Tue; Tuesday; T
F	Week-of-month	Number	3
a	am-pm-of-day	Text	PM
h	Clock-hour-of-am-pm (1-12)	Number	12
K	Hour-of-am-pm (0-11)	Number	0
k	Clock-hour-of-am-pm (1-24)	Number	0
H	Hour-of-day (0-23)	Number	0
m	Minute-of-hour	Number	30
s	Second-of-minute	Number	55
S	Fraction-of-second	Fraction	978
A	Milli-of-day	Number	1234
n	Nano-of-second	Number	987654321
N	Nano-of-day	Number	1234000000
V	Time-zone ID	Zone-id	America/Los_Angeles; Z; -08:30
z	Time-zone name	Zone-name	Pacific Standard Time; PST
X	Zone-offset Z for zero	Offset-X	Z; -08; -0830; -08:30; -083015; -08:30:15;
x	Zone-offset	Offset-x	+0000; -08; -0830; -08:30; -083015; -08:30:15;
Z	Zone-offset	Offset-Z	+0000; -0800; -08:00;
O	Localized zone-offset	Offset-O	GMT+8; GMT+08:00; UTC-08:00;
p	Pad next	Pad modifier	1

Example: converting in both directions between strings and dates using `DateTimeFormatter`.

```

import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

public class DateTime {

    public static void main(String[] args) {

        // Format a date with slashes instead of dashes
        DateTimeFormatter df =
            DateTimeFormatter.ofPattern("dd/MM/yyyy");
    }
}
```

```

System.out.println(df.format(LocalDate.now()));

// Parse a String to a date using the same(i.e., df) formatter
System.out.println(LocalDate.parse("01/04/2020", df));

// Format a Date and Time without timezone information
DateTimeFormatter nTZ =
    DateTimeFormatter.ofPattern("dd MMMM, yyyy h:mm a");
System.out.println(ZonedDateTime.now().format(nTZ));
}

}

```

Output:

2020/03/08
 2014-04-01
 8 March, 2020 10:10 PM

Converting Among Dates/Times, YMDHMS, and Epoch Seconds

- JAVA features a method called
 - *System.currentTimeMillis()*, presenting Epoch seconds with millisecond accuracy.
 - The “Epoch” is the beginning of time as far as modern operating systems (1st January 1970)

```

import java.util.*;

public class DateTime {

    public static void main(String[] args) {

        long end = System.currentTimeMillis();
        System.out.println("Epoch Time : " + end);
    }
}

```

Output:

Epoch Time : 1583690229071

Example: Convert a number of Seconds since the Epoch, to a local date/time

- The `java.time.Instant.ofEpochSecond(long epochSecond)` method obtains an instance of Instant using seconds from the epoch of 1970-01-01T00:00:00Z.

```
import java.time.Instant;

public class DateTime {

    public static void main(String[] args) {

        // Convert a number of Seconds since the Epoch, to a local date/time
        Instant epochSec = Instant.ofEpochSecond(1000000000L);
        System.out.println("Date and Time"+ epochSec);
    }
}
```

Output:

Date and Time: 2001-09-09T01:46:40Z

Example: Epoch-related numbers can be converted into, or obtained from, a local date/time.

- Java provides the Date class available in `java.util` package, this class encapsulates the current date and time.
- `Date()` constructor initializes the object with the current date and time.
- The `toInstant()` method of Date class in Java is used to convert a Date object to an Instant object.
- `getInstance()` method Creating a Calendar object.
- The `ofInstant(Instant instant, ZoneId zone)` method of `LocalDate` class in Java is used to create an instance of `LocalDate` from an Instant and zone ID. These two parameters are passed to the method and on the basis of those, an instance of LocalDate is created.

```

import java.time.Instant;
import java.time.ZoneId;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

public class DateTime {

    public static void main(String[] args) {

        // Convert a number of Seconds since the Epoch, to a local date/time
        Instant epochSec = Instant.ofEpochSecond(1000000000L);
        System.out.println(epochSec);

        //ZoneId.systemDefault() return the system default time-zone
        ZoneId zId = ZoneId.systemDefault();
        ZonedDateTime then = ZonedDateTime.ofInstant(epochSec, zId);
        System.out.println("The epoch was a billion seconds old on " + then);

        // Convert a date/time to Epoch seconds
        long epochSecond = ZonedDateTime.now().toInstant().getEpochSecond();
        System.out.println("Current epoch seconds = " + epochSecond);

        LocalDateTime now = LocalDateTime.now();
        ZonedDateTime there = now.atZone(ZoneId.of("UTC+05:30"));
        System.out.printf("When it's %s here, it's %s in Vancouver%n", now, there);

    }
}

```

Output:

2001-09-09T01:46:40Z
 The epoch was a billion seconds old on 2001-09-09T07:16:40+05:30[Asia/Calcutta]
 Current epoch seconds = 1583691395
 When it's 2020-03-08T23:46:35.344 here, it's 2020-03-08T23:46:35.344+05:30[UTC+05:30]
 in Vancouver

Difference Between Two Dates

You need to compute the difference between two dates

```

import java . time . LocalDate ;
import java . time . Period ;

public class ex3{

```

```

public static void main ( String [] args ) {

    /** The date at the end of the last century */
    LocalDate endofCentury = LocalDate .of (2000 , 12, 31);
    LocalDate now = LocalDate . now ();

    Period diff = Period . between ( endofCentury , now );
    System .out. printf ( " The 21 st century (up to %s) is %s old %n" ,
                           now , diff );
    System .out. printf ( " The 21 st century is %d years , %d months
                           and %d days old" ,diff . getYears () ,
                           diff . getMonths () , diff . getDays ());
}
}

```

OUTPUT :

The 21 st century (up to 2019 -10 -25) is P18Y9M25D old
The 21 st century is 18 years , 9 months and 25 days old

Parsing Strings into Dates

- Use a parse() method to parse a string into an object of that class.
- For example, LocalDate.parse(String) returns a LocalDate object for the date given in the input String.

```

import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;

```

```

public class DateTime {

```

```

    public static void main (String[] args) {

```

```

        String armisticeDate = "1914-11-11";
    }
}
```

```

// Parse a String to a LocalDate
LocalDate aLD = LocalDate.parse(armisticeDate);
System.out.println("Date: " + aLD);

// Parse a String to a LocalDateTime
String armisticeDateTime = "1914-11-11T11:11";
LocalDateTime aLDT =
        LocalDateTime.parse(armisticeDateTime);
System.out.println("Date/Time: " + aLDT);

String anotherDate = "27 Jan 2011";

DateTimeFormatter dfp = DateTimeFormatter.ofPattern("dd
        MMM uuuu");

// Parse a String to a LocalDate using the same(i.e., dfp) formatter
LocalDate random = LocalDate.parse(anotherDate, dfp);
System.out.println(anotherDate + " parses as " + random);

// Parse LocalDate to a LocalDate using the same(i.e., dfp) formatter
System.out.println(aLD + " formats as " + dfp.format(aLD));

}
}

```

Output:

Date: 1914-11-11
 Date/Time: 1914-11-11T11:11
 27 Jan 2011 parses as 2011-01-27
 1914-11-11 formats as 11 Nov 1914

Adding to or Subtracting from a Date or Calendar

- add or subtract a fixed number from a date using plusDays() and minusDays() method.

import java.time.LocalDate;

```

public class DateTIme {

    public static void main(String[] args) {

        LocalDate date = LocalDate.now();
        System.out.println("Today date: "+date);

        LocalDate yesterday = date.minusDays(1);
        System.out.println("Yesterday date: "+yesterday);

        LocalDate tomorrow = date.plusDays(1);
        //LocalDate tomorrow = yesterday.plusDays(2);
        System.out.println("Tomorrow date: "+tomorrow);
    }
}

```

Output:

Today date: 2020-03-09
 Yesterday date: 2020-03-08
 Tomorrow date: 2020-03-10

Example: using **period class of ofDays()** method, **plus()** method and **minus()** method.

- java.time offers a Period class to represent a length of time, such as a number of days, or hours and minutes.
- The **ofDays()** method of **Period Class** is used to obtain a period from given number of Days as parameter. This parameter is accepted in the form of integer. This method returns a Period with the given number of days.
- LocalDate and friends offer **plus()** and **minus()** methods to add or subtract a Period or other time-related object.

```

import java.time.LocalDate;
import java.time.Period;

```

```

public class DateTime {

    public static void main(String[] args) {

        LocalDate current = LocalDate.now();
        Period p = Period.ofDays(700);

        LocalDate feature = current.plus(p);
        System.out.printf("Seven hundred days from %s is %s%n",
                           current, feature);

        LocalDate previous = current.minus(p);
        System.out.printf("Seven hundred days from %s is %s%n",
                           current, previous);
    }
}

```

Output:

Seven hundred days from 2020-03-09 is 2022-02-07

Seven hundred days from 2020-03-09 is 2018-04-09

Interfacing with Legacy Date and Calendar Classes

<i>Legacy Date/Time interchange</i>		
Legacy Class	Convert to legacy	Convert to modern
java.util.Date	date.from(Instant)	Date.toInstant()
java.util.Calendar	calendar.toInstant()	-
java.util.GregorianCalendar	GregorianCalendar. from(ZonedDateTime)	calendar.toZonedDateTime()
java.util.TimeZone	-	timeZone.toZoneId()
java.time.DateTimeFormatter	-	dateTimeFormatter.toFormat()

Example:

- Java provides the Date class available in **java.util** package, this class encapsulates the current date and time.

- **Date()** constructor initializes the object with the current date and time.
- The ***toInstant()*** method of Date class in Java is used to convert a Date object to an Instant object.
- ***getInstance()*** method Creating a Calendar object.
- The ***ofInstant(Instant instant, ZoneId zone)*** method of ***LocalDate*** class in Java is used to create an instance of ***LocalDate*** from an Instant and zone ID. These two parameters are passed to the method and on the basis of those, an instance of LocalDate is created.

```

import java.util.*;
import java.time.LocalDateTime;
import java.time.ZoneId;

public class DateTime {

    public static void main(String[] args) {

        Date legacyDate = new Date();
        System.out.println(legacyDate);

        LocalDateTime newDate =
            LocalDateTime.ofInstant(legacyDate.toInstant(),
                                   ZoneId.systemDefault());
        System.out.println(newDate);

        // getInstance() method Creating a Calendar object
        Calendar c = Calendar.getInstance();
        System.out.println(c);

        LocalDateTime newCal = LocalDateTime.ofInstant(c.toInstant(),
                                                       ZoneId.systemDefault());
        System.out.println(newCal);
    }
}

```

Output:

Wed Nov 25 11:28:20 IST 2020

2020-11-25T11:28:20.040

```
java.util.GregorianCalendar[time=1606283900126,areFieldsSet=true,areAllFieldsSet=true,lenient=true,zone=sun.util.calendar.ZoneInfo[id="Asia/Calcutta",offset=19800000,dstSavings=0,useDaylight=false,transitions=7,lastRule=null],firstDayOfWeek=1,minimalDaysInFirstWeek=1,ERA=1,YEAR=2020,MONTH=10,WEEK_OF_YEAR=48,WEEK_OF_MONTH=4,DAY_OF_MONTH=25,DAY_OF_YEAR=330,DAY_OF_WEEK=4,DAY_OF_WEEK_IN_MONTH=4,AM_PM=0,HOUR=11,HOUR_OF_DAY=11,MINUTE=28,SECOND=20,MILLIS_ECOND=126,ZONE_OFFSET=19800000,DST_OFFSET=0]
```

2020-11-25T11:28:20.126

Structuring Data with Java

Using Arrays for Data Structuring

- The items in an array are same type.
- Array is a linear collection of data.
- The arrays can be two types:
 - *Arrays of Primitive Types,*
 - such as ints, booleans, etc., the data is stored in the array.
 - *Arrays of Objects,*
 - a reference is stored in the array, so the normal rules of reference variables and casting apply.

Note: in particular that if the array is declared as Object[], object references of any type can be stored in it without casting, although a valid cast is required to take an Object reference out and use it as its original type.

Example Array of Primitive Type:

Write a program to create an array of integer and display it.

```
public class JavaDataStructure {  
  
    public static void main(String[] args) {  
  
        //int[] month; // declare a reference  
        //month = new int[12]; // construct it  
  
        //int[] month = new int[12]; // short form  
  
        // even shorter is this initializer form:  
        int[] month = {  
            31, 28, 31, 30,  
            31, 30, 31, 31,  
            30, 31, 30, 31,  
        };  
  
        System.out.println("Array Elements are: ");
```

```

        for(int i=0; i<month.length; i++) {
            System.out.print(month[i]+ " ");
        }
    }
}

```

Output:

Array Elements are:

31 28 31 30 31 30 31 31 30 31 30 31

Example Array of Objects:

Write a program to create an array of String object and display it.

```

import java.util.*;

public class JavaDataStructure {

    public static void main(String[] args) {

        //inline initialization
        String[] strArray1 = new String[] {"A","B","C"};
        String[] strArray2 = {"A","B","C"};

        //initialization after declaration
        String[] strArray3 = new String[3];
        strArray3[0] = "A";
        strArray3[1] = "B";
        strArray3[2] = "C";

        System.out.println(strArray1.equals(strArray2)); // false

        // Arrays.toString() method to convert String array to String.
        System.out.println(Arrays.toString(strArray1).equals(Arrays.toString(strArray2))); // true
    }
}

```

Output:

false

true

Resizing an Array

- The array filled up, when the data is larger than the array size.
 - reallocate a new, bigger array and copy the elements into it, then insert rest of data.

Example:

```
import java.util.*;  
  
public class JavaDataStructure {  
  
    public static void main(String[] args) {  
  
        int[] a = {1, 2, 3};  
  
        int GROW_FACTOR = 2;  
        System.out.println("Before Resize array length = " + a.length);  
  
        System.out.println("Initial Array elements are: ");  
        for (int i=0; i<a.length; i++)  
            System.out.print(a[i] + " ");  
  
        System.out.println();  
  
        a = (int[]) resizeArray(a, a.length * GROW_FACTOR );  
  
        a[3] = 4;  
        a[4] = 5;  
        a[5] = 6;  
  
        System.out.println("After Resize array length = " + a.length);  
  
        System.out.println("Final Array elements are: ");  
        for (int i=0; i<a.length; i++)  
            System.out.print(a[i] + " ");  
    }  
  
    private static Object resizeArray (Object oldArray, int newSize) {  
  
        //returns the length of the specified array object, as an int.  
        int oldSize = java.lang.reflect.Array.getLength(oldArray);
```

```

// find the element type/data type of oldArray
Class elementType = oldArray.getClass().getComponentType();

//creates a new array with the specified data type and length.
Object newArray =
    java.lang.reflect.Array.newInstance(elementType, newSize);

int preserveLength = Math.min(oldSize, newSize);

if (preserveLength > 0)
    //copies an array from the specified source array, beginning at the
    //specified position, to the specified position of the destination array.
    System.arraycopy(oldArray, 0, newArray, 0, preserveLength);

return newArray;
}
}

```

Output:

Before Resize array length = 3

Initial Array elements are:

1 2 3

After Resize array length = 6

Final Array elements are:

1 2 3 4 5 6

ArrayList

- ArrayList is liked as an Array, but more dynamic.
- ArrayList is a standard class from java.util
 - *import java.util.ArrayList;*
- To create an ArrayList
 - ArrayList al=new ArrayList();
 - or
 - List al=new ArrayList();
 - Because ArrayList implements List interface

Important methods of ArrayList	
Method signature	Usage
add(Object o)	Add the given element at the end
add(int i, Object o)	Insert the given element at the specified position
clear()	Remove all element references from the Collection
contains(Object o)	True if the List contains the given Object
get(int i)	Return the object reference at the specified position
indexOf(Object o)	Return the index where the given object is found, or -1
remove(Object o)	Remove an object by reference or by position
remove(int i)	
toArray()	Return an array containing the objects in the Collection

Example: Demonstration of ArrayList

```

import java.util.ArrayList;

public class ArrayList1 {

    public static void main(String[] args) {

        // Declaring ArrayList
        ArrayList arrList = new ArrayList();

        // Appending the new element at the end of the list
        arrList.add(10);
        arrList.add(20);
        arrList.add(30);
        arrList.add(40);
        arrList.add(50);

        // Printing elements
        System.out.println("The initial ArrayList is: " + arrList);

        // Remove element at index 3 and print
        arrList.remove(3);
        System.out.println("After remove element, ArrayList is: " + arrList);

        // Printing elements one by one
        for (int i=0; i<arrList.size(); i++)
            System.out.print(arrList.get(i)+" ");

        System.out.println();
    }
}
```

```

//List contains the given Object
boolean x = arrList.contains(30);
System.out.println("List contains the given Object : " + x);

//clear ArrayList
arrList.clear();
System.out.println("After clearing element, ArrayList is: " + arrList);
}

}

```

Output:

```

The initial ArrayList is: [10, 20, 30, 40, 50]
After remove element, ArrayList is: [10, 20, 30, 50]
10 20 30 50
List contains the given Object : true
After clearing element, ArrayList is: []

```

Example: Write a program to add 3 string object to an ArrayList and display it.

```

import java.util.ArrayList;

public class ArrayList1 {

    public static void main(String[] args) {

        ArrayList aL =new ArrayList();

        aL.add("ABC");
        aL.add("PQR");
        aL.add("XYZ");

        System.out.println("The List of Names as ");
        for(int i=0; i<aL.size(); i++){
            System.out.println(aL.get(i));
        }
    }
}

```

Output:

```

The List of Names as
ABC
PQR
XYZ

```

Example: Write a program to create an ArrayList and insert 5 integer in it and find its sum.

```
import java.util.ArrayList;

public class ArrayList1 {

    public static void main(String[] args) {

        // Declaring ArrayList
        ArrayList arrList = new ArrayList();

        // Appending the new element at the end of the list
        arrList.add(10);
        arrList.add(20);
        arrList.add(30);
        arrList.add(40);
        arrList.add(50);

        // Printing elements
        System.out.println("The initial ArrayList is: " + arrList);

        // int sum=0;
        // for(int i=0; i<arrList.size(); i++){
        //     sum=sum + arrList.get(i); // give compilation error why?
        // }

        int sum=0;
        for(int i=0; i<arrList.size(); i++) {
            String s = arrList.get(i).toString();
            sum=sum+Integer.parseInt(s);
        }
        System.out.println("sum=" + sum);
    }
}
```

Output:

The initial ArrayList is: [10, 20, 30, 40, 50]
sum=150

Create ArrayList using Generic Collections

- In Generic Types mechanism, declare the ArrayList with a “type parameter”.
- The type parameter name appears in angle brackets after the declaration and instantiation.
- For example, to declare an ArrayList for holding String object references:
 - `ArrayList<String> data = new ArrayList<String>();`

Example:

```
import java.util.ArrayList;

public class ArrayList1 {

    public static void main(String[] args) {

        ArrayList<String> data = new ArrayList<String>();
        data.add("hello");
        data.add("goodbye");
        data.forEach(s -> System.out.println(s));
    }
}
```

Output:

```
hello
goodbye
```

Avoid Casting by Using Generics

- If the container classes define using the Generic Type mechanism, then casting can be avoided.

Example: Write a program to create ArrayList of integer type and find the sum of elements present in the list (without casting).

```
import java.util.ArrayList;

public class ArrayList1 {
```

```

public static void main(String[] args) {

    ArrayList<Integer> aL = new ArrayList<Integer>();
    aL.add(10);
    aL.add(20);
    aL.add(30);
    aL.add(40);
    aL.add(50);

    // Printing elements
    System.out.println("The ArrayList is: " + aL);

    int sum=0;
    for(int i=0; i<aL.size(); i++) {
        sum=sum+aL.get(i);
    }
    System.out.println("sum=" + sum);
}
}

```

Output:

The ArrayList is: [10, 20, 30, 40, 50]
sum=150

How Shall I Iterate

- Java provides many ways to iterate over collections of data. In newest-first order:
 - Iterable.forEach method (Java 8)
 - Java “foreach” loop (Java 5)
 - java.util.Iterator (Java 2)
 - Three-part for loop
 - “while” loop
 - Enumeration

Example:

```

import java.util.ArrayList;
import java.util.Iterator;
import java.utilEnumeration;
import java.util.*;

```

```
public class ArrayList1 {  
  
    public static void main(String[] args) {  
  
        ArrayList<String> arrList = new ArrayList<String>();  
        arrList.add("One");  
        arrList.add("Two");  
        arrList.add("Three");  
  
        System.out.println("Iterable.forEach method (Java 8) : " );  
        arrList.forEach(s -> System.out.println(s));  
  
        System.out.println("Java “foreach” loop (Java 5) : " );  
        for (String name : arrList) {  
            System.out.println(name);  
        }  
  
        System.out.println("Three-part for loop : " );  
        for(int i=0; i<arrList.size(); i++) {  
            System.out.println(arrList.get(i));  
        }  
  
        System.out.println("while loop : " );  
        Iterator<String> iter = arrList.iterator();  
        while(iter.hasNext()) {  
            System.out.println(iter.next());  
        }  
  
        System.out.println("java.util.Iterator (Java 2) : " );  
        Iterator it = arrList.iterator();  
        while (it.hasNext()) {  
            Object o = it.next();  
            System.out.println(o);  
        }  
  
        // creating object of type Enumeration<String>  
        Enumeration<String> e = Collections.enumeration(arrList);  
  
        System.out.println("Enumeration over list: " );  
        while (e.hasMoreElements())  
            System.out.println("Value is: " + e.nextElement());  
    }  
}
```

Output:

Iterable.forEach method (Java 8) :

One

Two

Three

Java “foreach” loop (Java 5) :

One

Two

Three

Three-part for loop :

One

Two

Three

while loop :

One

Two

Three

java.util.Iterator (Java 2) :

One

Two

Three

Enumeration over list :

Value is: One

Value is: Two

Value is: Three

$$A = \{10, 20, 50, 10, 40\} = \{10, 20, 50, 40\}$$

Eschewing Duplicates with a Set

- Set is an interface which extends Collection. It is an unordered collection of objects in which duplicate values cannot be stored.
- The difference between list and set is, a list can contain duplicate elements whereas Set contains unique elements only.
- Basically, Set is implemented by **HashSet**.
- Java HashSet class is used to create a collection that uses a hash table for storage. It inherits the AbstractSet class and implements Set interface.

- The important points about Java HashSet class are:
 - HashSet stores the elements by using a mechanism called **hashing**.
 - HashSet contains **unique** elements only.
 - HashSet **allows null value**.
 - HashSet class is non synchronized.
 - HashSet **doesn't maintain the insertion order**. Here, elements are inserted on the basis of their hashcode.
 - HashSet is the best approach for search operations.

Example: Simple Hashset example.

```
import java.util.*;

public class JavaSet {

    public static void main(String[] args) {

        //Creating HashSet and adding elements
        HashSet<String> A = new HashSet<String>();
        A.add("One");
        A.add("Two");
        A.add("Three");
        A.add("Four");
        A.add("Three"); //added duplicate elements.
        A.add("Five");

        //iterate over element of set,
        //HashSet doesn't allow duplicate elements.
        //HashSet doesn't maintain the insertion order.
        Iterator<String> it = A.iterator();
        while(it.hasNext()) {
            System.out.println(it.next());
        }
    }
}
```

Output:

Five
One
Four
Two
Three

Example: Different ways to remove an element from HashSet.

```
import java.util.*;  
  
public class JavaSet {  
  
    public static void main(String args[]){  
  
        HashSet<String> A = new HashSet<String>();  
        A.add("One");  
        A.add("Two");  
        A.add("Three");  
        A.add("Four");  
        A.add("Five");  
  
        //print the set  
        System.out.println("An initial list of elements: " + A);  
  
        //Removing specific element from HashSet  
        A.remove("Three");  
        System.out.println("After invoking remove(object) method: " + A);  
  
        //Removing elements on the basis of specified condition  
        A.removeIf(str->str.contains("Four"));  
        System.out.println("After invoking removeIf() method: " + A);  
  
        //Removing all the elements available in the set  
        A.clear();  
        System.out.println("After invoking clear() method: " + A);  
  
    }  
}
```

Output:

An initial list of elements: [Five, One, Four, Two, Three]
After invoking remove(object) method: [Five, One, Four, Two]

After invoking removeIf() method: [Five, One, Two]

After invoking clear() method: []

Example: Java code for demonstrating union, intersection and difference on Sets.

```
import java.util.*;
public class JavaSet {
    public static void main(String args[]){
        //Define the set A
        HashSet<String> A = new HashSet<String>();
        A.add("One");
        A.add("Two");
        A.add("Three");
        A.add("Four");
        A.add("Five");
        //print the set A
        System.out.println("Set A elements: " + A);

        //Define the set B
        HashSet<String> B = new HashSet<String>();
        B.add("Four");
        B.add("Five");
        B.add("Seven");
        //print the set B
        System.out.println("Set B elements: " + B);

        //To find Union between two sets
        HashSet<String> C = new HashSet<String>(A);
        C.addAll(B);
        System.out.println("After Union between Sets A and B : " + C);

        // To find intersection between two sets
        HashSet<String> D = new HashSet<String>(A);
        D.retainAll(B);
        System.out.println("After intersection between Sets A and B:" + D);

        //To find difference between two sets
        A.removeAll(B);
        System.out.println("After Difference between Sets A and B: " + A);
    }
}
```

Output:

Set A elements: [Five, One, Four, Two, Three]

Set B elements: [Five, Four, Seven]

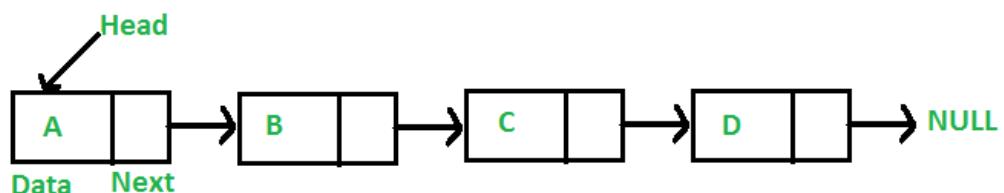
After Union between Sets A and B : [Five, One, Four, Seven, Two, Three]

After intersection between Sets A and B : [Five, Four]

After Difference between Sets A and B : [One, Two, Three]

Structuring Data in a Linked List

- A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations.
- In simple words, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.
- The elements in a linked list are linked using pointers as shown in the below image:
 - **Head** of the LinkedList only contains the Address of the **First element** of the List.
 - The Last element of the LinkedList contains **null** in the pointer part of the node because it is the end of the List so it doesn't point to anything as shown in the diagram.



Example: Java program to implement a Singly Linked List.

```
import java.io.*;
```

```
public class LinkedList {
```

```
    Node head; // head of list
```

```

// Linked list Node class. It is made static so that main() can access it
static class Node {

    int data;
    Node next;

    // Constructor
    Node(int d)
    {
        data = d;
        next = null;
    }
}

// *****INSERTION*****

// Method to insert a new node
public static LinkedList insert(LinkedList list, int data)
{
    // Create a new node with given data
    Node new_node = new Node(data);
    new_node.next = null;

    // If the Linked List is empty,
    // then make the new node as head
    if (list.head == null) {
        list.head = new_node;
    }
    else {
        // Else traverse till the last node
        // and insert the new_node there
        Node last = list.head;
        while (last.next != null) {
            last = last.next;
        }

        // Insert the new_node at last node
        last.next = new_node;
    }

    // Return the list by head
    return list;
}

```

```

// *****TRAVERSAL*****

// Method to print the LinkedList.
public static void printList(LinkedList list)
{
    Node currNode = list.head;

    System.out.print("\nLinkedList: ");

    // Traverse through the LinkedList
    while (currNode != null) {
        // Print the data at current node
        System.out.print(currNode.data + " ");

        // Go to next node
        currNode = currNode.next;
    }
    System.out.println("\n");
}

// *****DELETION BY KEY*****


// Method to delete a node in the LinkedList by KEY
public static LinkedList deleteByKey(LinkedList list, intkey)
{
    // Store head node
    Node currNode = list.head, prev = null;

    //
    // CASE 1:
    // If head node itself holds the key to be deleted

    if (currNode != null && currNode.data == key) {
        list.head = currNode.next; // Changed head

        // Display the message
        System.out.println(key + " found and deleted");

        // Return the updated List
        returnlist;
    }
}

```

```

// CASE 2:
// If the key is somewhere other than at head
// Search for the key to be deleted,
// keep track of the previous node
// as it is needed to change currNode.next
while (currNode != null&&currNode.data != key) {
    // If currNode does not hold key
    // continue to next node
    prev = currNode;
    currNode = currNode.next;
}

// If the key was present, it should be at currNode
// Therefore the currNode shall not be null
if (currNode != null) {
    // Since the key is at currNode
    // Unlink currNode from linked list
    prev.next = currNode.next;

    // Display the message
    System.out.println(key + " found and deleted");
}

// CASE 3: The key is not present
// If key was not present in linked list
// currNode should be null
if (currNode == null) {
    // Display the message
    System.out.println(key + " not found");
}

// return the List
returnlist;
}

// *****DELETION AT A POSITION*****

// Method to delete a node in the LinkedList by POSITION
public static LinkedList deleteAtPosition(LinkedList list, intindex)
{
    // Store head node

```

```

Node currNode = list.head, prev = null;

//
// CASE 1:
// If index is 0, then head node itself is to be deleted

if (index == 0 && currNode != null) {
    list.head = currNode.next; // Changed head

    // Display the message
    System.out.println(index + " position element deleted");

    // Return the updated List
    return list;
}

//
// CASE 2:
// If the index is greater than 0 but less than the size of LinkedList

// The counter
int counter = 0;

// Count for the index to be deleted,
// keep track of the previous node
// as it is needed to change currNode.next
while (currNode != null) {

    if (counter == index) {
        // Since the currNode is the required position
        // Unlink currNode from linked list
        prev.next = currNode.next;

        // Display the message
        System.out.println(index + " position element
                           deleted");
        break;
    }
    else {
        // If current position is not the index
        // continue to next node
        prev = currNode;
        currNode = currNode.next;
    }
}

```

```

        counter++;
    }
}

// If the position element was found, it should be at currNode
// Therefore the currNode shall not be null
//
// CASE 3: The index is greater than the size of the LinkedList
// In this case, the currNode should be null
if (currNode == null) {
    // Display the message
    System.out.println(index + " position element not found");
}

// return the List
returnlist;
}

// *****MAIN METHOD*****


// method to create a Singly linked list with n nodes
public static void main(String[] args)
{
    /* Start with the empty list. */
    LinkedList list = new LinkedList();

    //
    // *****INSERTION*****
    //

    // Insert the values
    list = insert(list, 1);
    list = insert(list, 2);
    list = insert(list, 3);
    list = insert(list, 4);
    list = insert(list, 5);
    list = insert(list, 6);
    list = insert(list, 7);
    list = insert(list, 8);

    // Print the LinkedList
    printList(list);
}

```

```
// *****DELETION BY KEY*****  
  
// Delete node with value 1  
// In this case the key is ***at head***  
deleteByKey(list, 1);  
  
// Print the LinkedList  
printList(list);  
  
// Delete node with value 4  
// In this case the key is present ***in the middle***  
deleteByKey(list, 4);  
  
// Print the LinkedList  
printList(list);  
  
// Delete node with value 10  
// In this case the key is ***not present***  
deleteByKey(list, 10);  
  
// Print the LinkedList  
printList(list);  
  
// *****DELETION AT POSITION*****  
  
// Delete node at position 0  
// In this case the key is ***at head***  
deleteAtPosition(list, 0);  
  
// Print the LinkedList  
printList(list);  
  
// Delete node at position 2  
// In this case the key is present ***in the middle***  
deleteAtPosition(list, 2);  
  
// Print the LinkedList  
printList(list);  
  
// Delete node at position 10  
// In this case the key is ***not present***  
deleteAtPosition(list, 10);
```

```
// Print the LinkedList  
printList(list);  
}  
}
```

Output:

LinkedList: 1 2 3 4 5 6 7 8

1 found and deleted

LinkedList: 2 3 4 5 6 7 8

4 found and deleted

LinkedList: 2 3 5 6 7 8

10 not found

LinkedList: 2 3 5 6 7 8

0 position element deleted

LinkedList: 3 5 6 7 8

2 position element deleted

LinkedList: 3 5 7 8

10 position element not found

LinkedList: 3 5 7 8

Example: Another example of Java program to implement a Singly Linked List.

```
public class LinkList {  
  
    public static void main(String[] args) {  
  
        System.out.println("Here is a demo of a Linked List in Java");  
    }  
}
```

```

LinkedList l = new LinkedList( ); //Constructor
l.add(new Object( ));
l.add("Hello");
l.add("World");
l.add("Java");
System.out.println("Here is a list of all the elements");
l.print( );

if (l.lookup("Hello"))
    System.err.println("Lookup works");
else
    System.err.println("Lookup does not work");
}

/* A TNode stores one node or item in the linked list. */
class TNode {
    TNode next;
    Object data;
    TNode(Object o) {
        data = o;
        next = null;
    }
}

protected TNode root;
protected TNode last;

/** Construct a LinkedList: initialize the root and last nodes. */
LinkedList( ) {
    root = new TNode(this);
    last = root;
}

/** Add one object to the end of the list. Update the "next"
 * reference in the previous end, to refer to the new node.
 * Update "last" to refer to the new node. */
void add(Object o) {
    last.next = new TNode(o);
    last = last.next;
}

public boolean lookup(Object o) {
    for (TNode p=root.next; p != null; p = p.next)

```

```

        if (p.data==o || p.data.equals(o))
            return true;
        return false;
    }

//linked list traversing through forward and print the element
void print( ) {
    for (TNode p=root.next; p != null; p = p.next)
        System.out.println("TNode" + p + " = " + p.data);
}
}

```

Output:

Here is a demo of a Linked List in Java

Here is a list of all the elements

TNodeLinkedList\$TNode@36ba530c = java.lang.Object@7a81197d

TNodeLinkedList\$TNode@5ca881b5 = Hello

TNodeLinkedList\$TNode@24d46ca6 = World

TNodeLinkedList\$TNode@4517d9a3 = Java

Lookup works

Example: Java program to implement a Singly Linked List, uses the existing class `java.util.LinkedList`

```

import java.util.LinkedList;
import java.util.ListIterator;

public class LinkList {

    public static void main(String args[]){

        System.out.println("Here is a demo of Java's LinkedList class");
        LinkedList<String> l = new LinkedList<>();
        l.add(new Object().toString());
        l.add("Hello");
        l.add("end of the list");
        l.add("World");
        l.add("Java");
        System.out.println("Here is a list of all the elements");

        ListIterator li = l.listIterator(0);

        //linked list traversing through forward
        System.out.println("linked list traversing through forward.");
    }
}

```

```

while (li.hasNext())
    System.out.println("Next to: " + li.next());

if (l.indexOf("Hello") < 0)
    System.err.println("Lookup does not work");
else
    System.err.println("Lookup works");

//linked list traversing through backwards.
System.out.println("linked list traversing through backwards.");
while (li.hasPrevious()) {
    System.out.println("Back to: " + li.previous());
}
}
}

```

Output:

Here is a demo of Java's LinkedList class

Here is a list of all the elements

linked list traversing through forward.

Next to: java.lang.Object@4617c264

Next to: Hello

Next to: end of the list

Next to: World

Next to: Java

Lookup works

linked list traversing through backwards.

Back to: Java

Back to: World

Back to: end of the list

Back to: Hello

Back to: [java.lang.Object@4617c264](#)

Example: Java program to implement a Singly Linked List, **adding elements to Linked List** using add(), addFirst() and addLast() methods to add the elements at the desired locations in the Linked List. Similarly deleting **elements from Linked List** using remove(), removeFirst() and removeLast() methods to remove the elements at the desired locations in the Linked List.

```
import java.util.*;
```

```
public class JavaSet {  
  
    public static void main(String args[]){  
  
        LinkedList<String> list=new LinkedList<String>();  
  
        //Adding elements to the Linked list  
        list.add("Steve");  
        list.add("Carl");  
        list.add("Raj");  
  
        //Iterating LinkedList  
        Iterator<String> iterator = list.iterator();  
  
        //print the element of linked list  
        System.out.println("Initial Linked List:");  
        while(iterator.hasNext()){  
            System.out.println(iterator.next());  
        }  
  
        //Adding an element to the first position  
        list.addFirst("Negan");  
  
        //Adding an element to the last position  
        list.addLast("Rick");  
  
        //Adding an element to the 3rd position  
        list.add(2, "Glenn");  
  
        //Iterating LinkedList  
        Iterator<String> iterator1=list.iterator();  
        //Iterating LinkedList and print the elements  
        System.out.println("After Adding 3 element to Linked List:");  
        while(iterator1.hasNext()){  
            System.out.println(iterator1.next());  
        }  
  
        //Removing First element  
        //Same as list.remove(0);  
        list.removeFirst();  
  
        //Removing Last element  
        list.removeLast();
```

```

//removing 2nd element, index starts with 0
list.remove(1);

//Iterating LinkedList
Iterator<String> iterator2=list.iterator();
//Iterating LinkedList
System.out.println("After delete elements from Linked List:");
while(iterator2.hasNext()){
    System.out.println(iterator2.next()+" ");
}
}

```

Output:

Initial Linked List:

Steve
Carl
Raj

After Adding 3 element to Linked List:

Negan
Steve
Glenn
Carl
Raj
Rick

After delete elements from Linked List:

Steve
Carl
Raj

Mapping with Hashtable and HashMap

Java Hashtable class

- Java Hashtable class implements a hashtable, which maps keys to values.
 - A Hashtable is an array of a list. Each list is known as a bucket. The position of the bucket is identified by calling the hashCode() method. A Hashtable contains values based on the key.
 - Java Hashtable class contains unique elements.


```

//putIfAbsent(K,V), inserts, as the specified pair is unique
ht.putIfAbsent(105,"Gaurav");
System.out.println("Updated HashTable: "+ ht);

//Returns the current value, as the specified pair already exist
ht.putIfAbsent(101,"Jay");
System.out.println("Updated Hash Table: " + ht);
}
}

```

Output:

```

Key: 104 Value: Amit
Key: 103 Value: Rahul
Key: 102 Value: Ravi
Key: 101 Value: Vijay
Key: 100 Value: Amit
Before remove: {104=Amit, 103=Rahul, 102=Ravi, 101=Vijay, 100=Amit}
After remove: {104=Amit, 103=Rahul, 101=Vijay, 100=Amit}
Vijay
Not Found
Updated HashTable: {105=Gaurav, 104=Amit, 103=Rahul, 101=Vijay, 100=Amit}

Updated Hash Table: {105=Gaurav, 104=Amit, 103=Rahul, 101=Vijay, 100=Amit}

```

Java HashMap class

- Java HashMap class implements the map interface by using a hash table.
 - Java HashMap class contains values based on the key.
 - Java HashMap class contains only unique keys.
 - Java HashMap class may have one null key and multiple null values.
 - Java HashMap class is non synchronized.
 - Java HashMap class maintains no order.

Example:

```
import java.util.*;

public class HashMap1 {

    public static void main(String[] args) {

        // creating a hash map
        HashMap<Integer, String> hm = new HashMap<Integer, String>();

        System.out.println("Initial list of elements: " + hm); //empty list

        hm.put(100, "Amit");
        hm.put(101, "Vijay");
        hm.put(102, "Rahul");

        System.out.println("After invoking put() method ");
        //print the key and its corresponding value in hash table
        for(Map.Entry m : hm.entrySet()){
            System.out.println(m.getKey() + " " + m.getValue());
        }

        //putIfAbsent(K,V), inserts, as the specified pair is unique
        hm.putIfAbsent(103, "Gaurav");

        System.out.println("After invoking putIfAbsent() method ");
        for(Map.Entry m : hm.entrySet()){
            System.out.println(m.getKey() + " " + m.getValue());
        }

        //print the has map
        System.out.println("Initial list of elements: " + hm);

        //value-based removal
        hm.remove(101);
        System.out.println("Updated list of elements: " + hm);

        //key-value pair based removal
        hm.remove(102, "Rahul");
        System.out.println("Updated list of elements: " + hm);
    }
}
```

Output:

Initial list of elements: {} //empty list

After invoking put() method

100 Amit

101 Vijay

102 Rahul

After invoking putIfAbsent() method

100 Amit

101 Vijay

102 Rahul

103 Gaurav

Initial list of elements: {100=Amit, 101=Vijay, 102=Rahul, 103=Gaurav}

Updated list of elements: {100=Amit, 102=Rahul, 103=Gaurav}

Updated list of elements: {100=Amit, 103=Gaurav}

Example: Another example of hash map

```
import java.util.*;

public class HashMap1 {

    public static void main(String[] args) {

        // creating a hash table
        Map<String, String> map = new HashMap<String, String>();

        // The hash maps from company name to address.
        // In real life this might map to an Address object...
        map.put("Adobe", "Mountain View, CA");
        map.put("IBM", "White Plains, NY");
        map.put("Learning Tree", "Los Angeles, CA");
        map.put("Microsoft", "Redmond, WA");
        map.put("Netscape", "Mountain View, CA");
        map.put("O'Reilly", "Sebastopol, CA");
        map.put("Sun", "Mountain View, CA");

        // Two versions of the "retrieval" phase.
        // Version 1: get one pair's value given its key
        // (presumably the key would really come from user input):
        String queryString = "O'Reilly";
        System.out.println("You asked about " + queryString + ".");
    }
}
```

```

String resultString = map.get(queryString);
System.out.println("They are located in: " + resultString);
System.out.println();

// Version 2: get ALL the keys and values
// (maybe to print a report, or to save to disk)
for( String key : map.keySet() ) {
    System.out.println("Key " + key + "; Value " +
                       map.get(key));
}

System.out.println();

// Version 3: print all the map entry
map.entrySet().forEach(mE ->
    System.out.println("Key + " + mE.getKey() + "; Value "
                       +mE.getValue()));

System.out.println();

// Version 2: get ALL the keys and values
// with removing Company name Sun and its Address
Iterator<String> it = map.keySet( ).iterator( );
while (it.hasNext( )) {
    String key = it.next();
    if (key.equals("Sun")) {
        it.remove();
        continue;
    }
    System.out.println("Company " + key + ";" + "Address " +
                       map.get(key));
}
}
}

```

Output:

You asked about O'Reilly.
 They are located in: Sebastopol, CA

Key IBM; Value White Plains, NY

Key Learning Tree; Value Los Angeles, CA

Key O'Reilly; Value Sebastopol, CA

Key Microsoft; Value Redmond, WA

Key Adobe; Value Mountain View, CA

Key Sun; Value Mountain View, CA

Key Netscape; Value Mountain View, CA

Key + IBM; Value White Plains, NY

Key + Learning Tree; Value Los Angeles, CA

Key + O'Reilly; Value Sebastopol, CA

Key + Microsoft; Value Redmond, WA

Key + Adobe; Value Mountain View, CA

Key + Sun; Value Mountain View, CA

Key + Netscape; Value Mountain View, CA

Company IBM; Address White Plains, NY

Company Learning Tree; Address Los Angeles, CA

Company O'Reilly; Address Sebastopol, CA

Company Microsoft; Address Redmond, WA

Company Adobe; Address Mountain View, CA

Company Netscape; Address Mountain View, CA

Sorting a Collection

- If the data into a collection in random order, and now you want to sorted.
 - If data is in an array with unsorted manner, then sort it using the static sort() method of the Arrays utility class.
 - `Arrays.sort()` in Java
 - If data is in a Collection, then use the static sort() method of the Collections class.
 - `Collections.sort()` in Java

Arrays.sort() in Java

- `sort()` method is a [java.util.Arrays](#) class method.

Example: A sample Java program to sort an array using Arrays.sort(). It by default sorts in ascending order.

```
import java.util.Arrays;

public class ArraySort {

    public static void main(String[] args) {

        //S contains 4 elements
        String[] S = {
            "painful",
            "mainly",
            "gaining",
            "raindrops"
        };

        //sorting the array of string in ascending order
        Arrays.sort(S);
        //Display
        System.out.println("Sorted String Array: ");
        for (int i=0; i<S.length; i++) {
            System.out.println(S[i]);
        }

        //arr contains 8 elements
        int[] arr = {13, 7, 6, 45, 21, 9, 101, 102};

        //sorting the array of integer in ascending order
        Arrays.sort(arr);
        //Convert the array of integer to String and Display
        System.out.printf("Sorted Integer Array: %s", Arrays.toString(arr));
    }
}
```

Output:

Sorted String Array:

gaining
mainly
painful
raindrops

Sorted Integer Array: [6, 7, 9, 13, 21, 45, 101, 102]

Example: A sample Java program to sort an array in descending order

```
import java.util.Arrays;
import java.util.Collections;

public class ArraySort {

    public static void main(String[] args) {

        //S contains 4 elements
        String[] S = {
            "painful",
            "mainly",
            "gaining",
            "raindrops"
        };

        //sorting the array of string in descending order
        Arrays.sort(S, Collections.reverseOrder());
        System.out.println("Sorted String Array: ");
        for (int i=0; i<S.length; i++) {
            System.out.println(S[i]);
        }

        //arr contains 8 elements
        Integer[] arr = {13, 7, 6, 45, 21, 9, 2, 100};

        //sorting the array of integer in descending order
        Arrays.sort(arr, Collections.reverseOrder());
        //Convert the array of integer to String and Display
        System.out.printf("Sorted Integer Array: %s", Arrays.toString(arr));
    }
}
```

Output:

Sorted String Array:
raindrops
painful
mainly

gaining

Sorted Integer Array: [100, 45, 21, 13, 9, 7, 6, 2] xyz

Collections.sort() in Java

- **java.util.Collections.sort()** method is present in `java.util.Collections` class. It is used to sort the elements present in the specified list of Collection in ascending order.
- It works similar to [java.util.Arrays.sort\(\)](#) method but it is better than as it can sort the elements of Array as well as linked list, queue and many more present in it.

Example: Java program to demonstrate working of `Collections.sort()` to ascending and descending order.

```
import java.util.ArrayList;
import java.util.Collections;

public class CollectionSort {

    public static void main(String[] args) {

        // Create a list of strings
        ArrayList<String> al = new ArrayList<String>();
        al.add("yellow");
        al.add("green");
        al.add("white");
        al.add("red");
        al.add("blue");

        //sorting the elements of ArrayList in ascending order.
        Collections.sort(al);
        System.out.println("Sorted List in Ascending Order:\n" + al);

        //sorting the elements of ArrayList in descending order.
        Collections.sort(al, Collections.reverseOrder());
        System.out.println("Sorted List in Descending Order:\n" + al);
    }
}
```

Output:

Sorted List in Ascending Order:

[blue, green, red, white, yellow]

Sorted List in Descending Order:

[yellow, white, red, green, blue]

Example: Java program to sort the array of string using compareTo().

```
import java.util.Comparator;

/* Comparator for comparing strings ignoring first character.*/
public class SubstringComparator implements Comparator<String> {
    @Override
    public int compare(String s1, String s2) {
        s1 = s1.substring(1);
        s2 = s2.substring(1);
        return s1.compareTo(s2);
        // or, more concisely:
        // return s1.substring(1).compareTo(s2.substring(1));
    }
}

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;

public class SubstrCompDemo {

    static void dump(String[] args, String title) {
        System.out.println(title);
        for (int i=0; i<args.length; i++)
            System.out.println(args[i]);
    }

    public static void main(String[] args) {

        String[] strings = {
            "painful",
            "mainly",
            "gaining",
            "raindrops"
        };
        Arrays.sort(strings);
        dump(strings, "Using Default Sort");
        Arrays.sort(strings, new SubstringComparator());
        dump(strings, "Using SubstringComparator");
    }
}
```

```
}
```

Output:

Using Default Sort

gaining
mainly
painful
raindrops

Using SubstringComparator

raindrops
painful
gaining
mainly

Avoiding the Urge to Sort

- Suppose data needs to be sorted
 - Oneway is insert the data and use the sort operation explicitly. Already we have discussed these methods.
 - Another way is keep the data in sorted manner at the time of insertions of data. So that the data are kept sorted at all times. For which use a data structure a TreeSet or a TreeMap.
 - a TreeSet (which keeps objects in order) or
 - a TreeMap (which keeps the keys in order and maps from keys to values; the keys would be the name and the values would be the Person objects).
 - Both insert the objects into a tree in the correct order, so an Iterator that traverses the tree always returns the objects in sorted order.

Java TreeSet class

- Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface.

- The objects of the TreeSet class are stored in ascending order.
- Java TreeSet class contains unique elements only like HashSet.
- Java TreeSet class access and retrieval times are quiet fast.
- Java TreeSet class doesn't allow null element.
- Java TreeSet class is non synchronized.
- Java TreeSet class maintains ascending order.

Example: A simple example of TreeSet

```

import java.util.Iterator;
import java.util.TreeSet;

public class JavaTreeSet {

    public static void main(String[] args) {

        // A TreeSet keeps objects in sorted order. Use a Comparator
        // published by String for case-insensitive sorting order.
        //Creating and adding elements
        TreeSet<String> TS = new
            TreeSet<>(String.CASE_INSENSITIVE_ORDER);
        TS.add("Gosling");
        TS.add("da Vinci");
        TS.add("van Gogh");
        TS.add("Java To Go");
        TS.add("Vanguard");
        TS.add("Darwin");
        TS.add("Darwin"); // TreeSet is Set, ignores duplicates.

        System.out.printf("Our set contains %d elements", TS.size());
        System.out.println();

        // Since it is sorted we can easily get first element
        System.out.println("Lowest (alphabetically) is " + TS.first());

        // Since it is sorted we can easily get last element
        System.out.println("Highest (alphabetically) is " + TS.last());
    }
}
```

```

System.out.println("Traversing element in ascending order: ");
Iterator<String> itr = TS.iterator();
while(itr.hasNext()){
    System.out.println(itr.next());
}

System.out.println("Traversing element in descending order: ");
Iterator itr1=TS.descendingIterator();
while(itr1.hasNext()){
    System.out.println(itr1.next());
}

//display ascending set
System.out.println("Initial Set: "+ TS);

//display descending set
System.out.println("Reverse Set: " + TS.descendingSet());

//headSet(), which give a new Set of objects of the same class,
//containing the objects lexically before a given value.
System.out.println("Head Set: " + TS.headSet("Java To Go", true));

//The tailSet() methods, return objects greater than a given value
System.out.println("Tail Set: " + TS.tailSet("Java To Go", true));

//subSet() method return a range
System.out.println("SubSet: " + TS.subSet("Darwin", true, "van
Gogh", true));
//print the number of elements higher than k
System.out.println(TS.tailSet("k").toArray().length +
                    " elements higher than \"k\"");

// Print the whole list in sorted order
System.out.println("Sorted list:");
TS.forEach(name -> System.out.println(name));
}
}

```

Output:

Our set contains 6 elements
 Lowest (alphabetically) is da Vinci

Lowest (alphabetically) is Vanguard

Traversing element in ascending order:

da Vinci

Darwin

Gosling

Java To Go

van Gogh

Vanguard

Traversing element in descending order:

Vanguard

van Gogh

Java To Go

Gosling

Darwin

da Vinci

Initial Set: [da Vinci, Darwin, Gosling, Java To Go, van Gogh, Vanguard]

Reverse Set: [Vanguard, van Gogh, Java To Go, Gosling, Darwin, da Vinci]

Head Set: [da Vinci, Darwin, Gosling, Java To Go]

Tail Set: [Java To Go, van Gogh, Vanguard]

SubSet: [Darwin, Gosling, Java To Go, van Gogh]

2 elements higher than "k"

Sorted list:

da Vinci

Darwin

Gosling

Java To Go

van Gogh

Vanguard

Java TreeMap class

- Java TreeMap class is a red-black tree based implementation. It provides an efficient means of storing key-value pairs in sorted order.
- Java TreeMap contains values based on the key. It implements the NavigableMap interface and extends AbstractMap class.
- Java TreeMap contains only unique elements.
- Java TreeMap cannot have a null key but can have multiple null values.
- Java TreeMap is non synchronized.

- Java TreeMap maintains ascending order.

Example: A simple example of TreeMap

```

import java.util.Map;
import java.util.TreeMap;

public class JavaTreeMap {

    public static void main(String[] args) {

        //Creating and adding elements
        TreeMap<Integer, String> TM = new TreeMap<Integer, String>();
        TM.put(100, "Gosling");
        TM.put(101, "da Vinci");
        TM.put(102, "van Gogh");
        TM.put(103, "Java To Go");
        TM.put(104, "Vanguard");
        TM.put(105, "Darwin");
        TM.put(105, "Darwin"); // TreeSet is Set, ignores duplicates.

        System.out.printf("Our set contains %d elements", TM.size());
        System.out.println();

        // Since it is sorted we can easily get first element
        System.out.println("Lowest key is " + TM.firstKey());

        // Since it is sorted we can easily get last element
        System.out.println("Highest Key is " + TM.lastKey());

        //Traversing TreeMap in ascending
        for(Map.Entry m : TM.entrySet()){
            System.out.println(m.getKey()+" "+m.getValue());
        }

        //display ascending set
        System.out.println("Initial Set: "+ TM);

        //display descending set
        System.out.println("Reverse Set: "+ TM.descendingKeySet());
    }
}
```

```

//Maintains descending order
System.out.println("descendingMap: "+TM.descendingMap());

//headMap(), which give a new Set of objects of the same class,
//containing the objects lexically before a given value.
System.out.println("Head Set: " + TM.headMap(103, true));

//The tailMap() methods, return objects greater than a given value
System.out.println("Tail Set: " + TM.tailMap(102, true));

//subMap() method return a range
System.out.println("SubSet: " + TM.subMap(101, true, 104, true));
}
}

```

Output:

Our set contains 6 elements

Lowest key is 100

Highest Key is 105

100 Gosling

101 da Vinci

102 van Gogh

103 Java To Go

104 Vanguard

105 Darwin

Initial Set: { 100=Gosling, 101=da Vinci, 102=van Gogh, 103=Java To Go, 104=Vanguard, 105=Darwin }

Reverse Set: [105, 104, 103, 102, 101, 100]

descendingMap: { 105=Darwin, 104=Vanguard, 103=Java To Go, 102=van Gogh, 101=da Vinci, 100=Gosling }

Head Set: { 100=Gosling, 101=da Vinci, 102=van Gogh, 103=Java To Go }

Tail Set: { 102=van Gogh, 103=Java To Go, 104=Vanguard, 105=Darwin }

SubSet: { 101=da Vinci, 102=van Gogh, 103=Java To Go, 104=Vanguard }

Finding an Object in a Collection

- Let a given collection contains the values and now need to find out an object of the given value.
- There is quite a variety of methods, depending on the collection class.

Method(s)	Meaning	Implementing classes
binarySearch()	Fairly fast search	Arrays, Collections
contains()	Search	ArrayList, HashSet, Hashtable, Link-List, Properties, Vector
containsKey(), containsValue()	Checks if the collection contains the object as a Key or as a Value	HashMap, Hashtable, Properties, TreeMap
indexOf()	Returns location where object is found	ArrayList, LinkedList, List, Stack, Vector
search()	Search	Stack

Java.util.Arrays.binarySearch() Method

- The `java.util.Arrays.binarySearch(Object[] a, Object key)` method searches the specified array for the specified object using the binary search algorithm.
 - `a` – This is the array to be searched.
 - `key` – This is the value to be searched for.
- The array be sorted into ascending order according to the natural ordering of its elements prior to making this call.
- If it is not sorted, the results are undefined.

Example: The following example shows the usage of `java.util.Arrays.binarySearch()` method.

```
import java.util.Arrays;

public class ArrayBinarySearch{

    public static void main(String[] args) {

        // initializing unsorted array
        Object arr[] = {10,2,22,69};

        // sorting array
        Arrays.sort(arr);
```

```

// let us print all the elements available
System.out.println("The sorted array is:");
for (Object number : arr) {
    System.out.println("Number = " + number);
}

// entering the value to be searched
int searchVal = 22;

// Arrays.binarySearch() method return the index value
int retVal = Arrays.binarySearch(arr, searchVal);

System.out.println("The index of element 22 is : " + retVal);
}
}

```

Output:

The sorted array is:

Number = 2

Number = 10

Number = 22

Number = 69

The index of element 22 is : 2

java.util.Collections.binarySearch() Method

- The *binarySearch(List<? extends Comparable<? super T>>, T)* method is used to search the specified list for the specified object using the binary search algorithm.
- Following is the declaration for `java.util.Collections.binarySearch()` method.
 - *public static <T> int binarySearch(List<? extends Comparable<? super T>> list, T key)*
 - list – This is the list to be searched.
 - key – This is the key to be searched for.

Example: The following example shows the usage of `java.util.Collections.binarySearch()`

```
import java.util.*;

public class CollectionsBinarySearch {

    public static void main(String args[]) {

        // create arraylist
        ArrayList<String> arlst = new ArrayList<String>();

        // populate the list
        arlst.add("ABCD");
        arlst.add("PROVIDES");
        arlst.add("QUALITY");
        arlst.add("TUTORIALS");
        System.out.println("The ArrayList elements are: ");

        // Display
        Iterator<String> itr= arlst.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }

        // search the list for key 'QUALITY'
        int index = Collections.binarySearch(arlst, "QUALITY");

        System.out.println("QUALITY is available at index: "+index);
    }
}
```

Output:

The ArrayList elements are:

ABCD
PROVIDES
QUALITY
TUTORIALS
'QUALITY' is available at index: 2

Example: This example plays a little game of “find the hidden number” (or “needle in a haystack”): the numbers to look through are stored in an array. As games go, it’s fairly pathetic: the computer plays against itself, so you probably know who’s going to win. It wrote that in such a way that the data array contains valid numbers. The array to be used with Arrays.binarySearch() must be in sorted order, but because we just filled it with random numbers, it isn’t initially sorted. Hence, call Arrays.sort() on the array. Then call Arrays.binarySearch(), passing in the array and the value to look for.

```
import java.util.Arrays;
import java.util.Random;

public class ArrayHunt {

    /** the maximum (and actual) number of random ints to allocate */
    protected final static int MAX = 4000;

    /** the value to look for */
    protected final static int NEEDLE = 1999;
    int[] haystack;
    Random r;

    public static void main(String[] argv) {
        ArrayHunt h = new ArrayHunt();
        if (argv.length == 0)
            h.play();
        else {
            intwon = 0;
            intgames = Integer.parseInt(argv[0]);
            for (int i=0; i<games; i++)
                if (h.play())
                    ++won;
            System.out.println("Computer won " + won + " out of " +
                               games + ".");
        }
    }

    /** Construct the hunting ground */
    public ArrayHunt() {
        haystack = new int[MAX];
        r = new Random();
    }
}
```

```

/** Play one game. */
public boolean play() {
    int i;
    // Fill the array with random data (hay?)
    for (i=0; i<MAX; i++) {
        haystack[i] = (int)(r.nextFloat() * MAX);
    }

    // Precondition for binary search is that data be sorted!
    Arrays.sort(haystack);

    // Look for needle in haystack
    i = Arrays.binarySearch(haystack, NEEDLE);
    if (i>= 0) { // Found it, we win.
        System.out.println("Value " + NEEDLE + " occurs at
                           haystack[" + i + "]");
        return true;
    }
    else { // Not found, we lose.
        System.out.println("Value " + NEEDLE +
                           " does not occur in haystack; nearest value is "+
                           haystack[-(i+2)] + " (found at " + -(i+2) + ")");
        return false;
    }
}

```

Output:

Value 1999 occurs at haystack[1977]

Converting a Collection to an Array

- If you have an ArrayList or other Collection and you need an array,
 - you can get it just by calling the Collection's `toArray()` method, with no arguments.
 - you get an array whose type is Object[].

Example: shows code for converting an ArrayList to an array of type Object.

```
import java.util.*;

public class CollectionToArray {

    public static void main(String args[]) {

        // create arraylist
        List<String> list = new ArrayList<>();
        list.add("Blobbo");
        list.add("Cracked");
        list.add("Dumbo");

        System.out.println("The ArrayList elements are: ");
        Iterator<String> itr= list.iterator();
        while(itr.hasNext()){
            System.out.println(itr.next());
        }

        // Convert a collection to Object[], which can store objects
        // of any type.
        Object[] ol = list.toArray();
        System.out.println("Array of Object has length " + ol.length);

        System.out.println("The Array Objects are: ");
        for(int i=0; i<ol.length; i++)
            System.out.println(ol[i]);

        String[] s = (String[]) list.toArray(new String[0]);
        System.out.println("Array of String has length " + s.length);

        System.out.println("The Array elements are: ");
        for(int i=0; i<s.length; i++)
            System.out.println(s[i]);
    }
}
```

Output:

The ArrayList elements are:

Blobbo

Cracked

Dumbo

Array of Object has length 3

The Array Objects are:

Blobbo

Cracked

Dumbo

Array of String has length 3

The Array elements are:

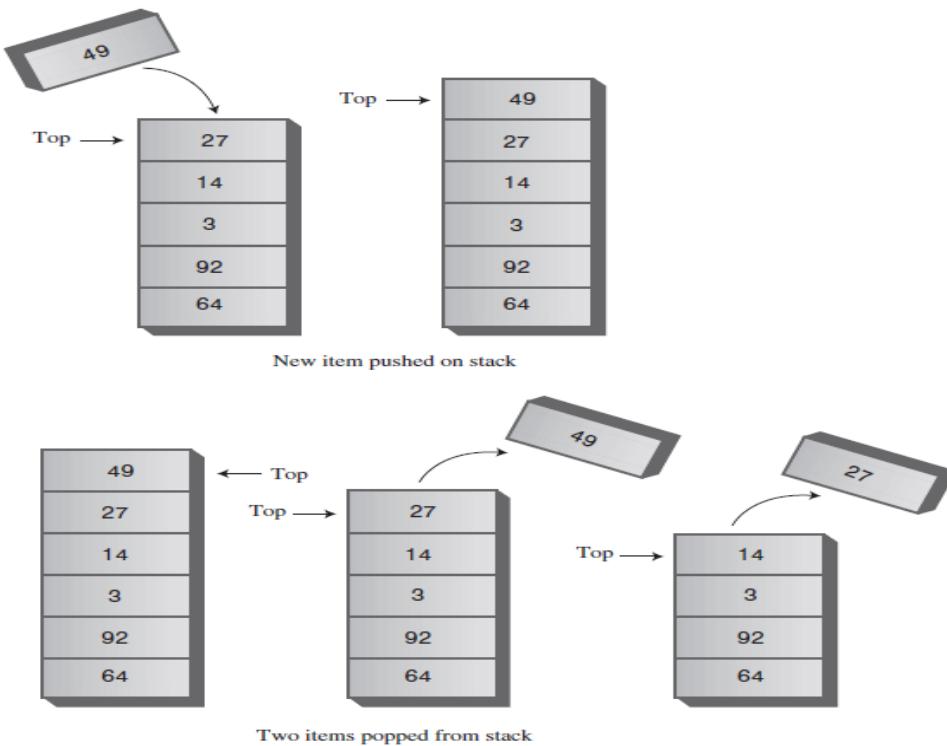
Blobbo

Cracked

Dumbo

Stack

- Stack is a list in which, insertion and deletion can be performed only at one end that is called **top**.
- Stack is a recursive data structure having pointer to its top element.
- Stacks are sometimes called as Last-In-First-Out (LIFO) lists i.e. the element which is inserted first in the stack, will be deleted last from the stack.
- The 3 basic operations of stack are
 - push() : add an element to stack
 - pop() : remove an element from stack
 - peek() : examine top element without removing
- apart from 3 basic operations, the other operations of stacks are
 - empty() : whether the stack is empty or not, if stack is empty then it return true otherwise return false. The top variable is at -1, if the stack is empty.
 - full() : whether the stack is empty or not, if stack is full then it return true otherwise return false. The top variable is at maxSize-1, if the stack is full.
 - search(): find the particular element present in the stack.



Example: Simple stack program using primitive data type.

```
import java.util.Scanner;

class StackX{

    private int maxSize; // size of stack array
    private long[] stackArray;
    private int top; // top of stack

    public StackX(int s) // constructor
    {
        maxSize = s; // set array size
        stackArray = new long[maxSize]; // create array
        top = -1; // no items yet
    }

    // put item on top of stack
    public void push(long j) {
        // increment top and insert item
        stackArray[++top] = j;
    }
}
```

```

// take item from top of stack
public long pop() {
    // access item, decrement top
    return stackArray[top--];
}

// peek at top of stack
public long peek() {
    return stackArray[top];
}

// true if stack is empty
Public boolean isEmpty() {
    return (top == -1);
}

Public boolean isFull() // true if stack is full
{
    return (top == maxSize-1);
}

} // end class StackX

```

```

public class StackDemo {

    public static void main(String[] args) {

        // make new stack
        StackX theStack = new StackX(5);

        while(!theStack.isFull()) {

            // Create a Scanner object
            Scanner myObj = new Scanner(System.in);

            System.out.println("Push the Item: ");
            // Read user input
            long invalue = myObj.nextLong();
            // push items onto stack
            theStack.push(invalue);
        }
    }
}

```

```

if(theStack.isFull())
    System.out.println("Now the stack is full.");
}

// Displaying element on the top of the stack
long element = theStack.peek();
System.out.println("Element on stack top : " + element);

System.out.println("The poped items are : ");
// delete item from stack, until it's empty,
while( !theStack.isEmpty() )
{
    long value = theStack.pop();
    System.out.print(value); // display it
    System.out.print(" ");
} // end while

System.out.println(" ");
System.out.println("Now the stack is empty.");
} // end main()
}// end StackDemo class

```

Output:

Push the Item:

20

Push the Item:

10

Push the Item:

80

Push the Item:

40

Push the Item:

30

Now the stack is full.

Element on stack top : 30

The poped items are :

30 40 80 10 20

Now the stack is empty.

Example: simple stack-based calculator using BigDecimal as its numeric data type.

```
import java.math.BigDecimal;
import java.util.Stack;

public class StackDemo {

    /** an array of Objects, simulating user input */
    public static Object[] testInput = {
        new BigDecimal("3419229223372036854775807.23343"),
        new BigDecimal("2.0"),
        "*",
    };

    public static void main(String[] args) {

        StackDemo calc = new StackDemo();
        System.out.println("The result is " + calc.calculate(testInput));
    }

    // Stack of numbers being used in the calculator.
    Stack<BigDecimal> stack = new Stack<>();

    // Calculate a set of operands; the input is an Object array containing
    // either BigDecimal objects (which may be pushed onto the Stack) and
    // operators (which are operated on immediately).
    // @param input
    // @return
    public BigDecimal calculate(Object[] input) {

        BigDecimal tmp;

        for (int i = 0; i < input.length; i++) {

            Object o = input[i];

            if (o instanceof BigDecimal) {
                stack.push((BigDecimal) o);
            }
        }

        elseif (o instanceof String) {
```

```

switch (((String)o.charAt(0)) {

    // + and * are commutative, order doesn't matter
    case '+':
        stack.push((stack.pop()).add(stack.pop()));
        break;

    case '*':
        stack.push((stack.pop()).multiply(stack.pop()));
        break;

    // - and /, order *does* matter
    case '-':
        tmp = (BigDecimal)stack.pop();
        stack.push((stack.pop()).subtract(tmp));
        break;

    case '/':
        tmp = stack.pop();
        stack.push((stack.pop()).divide(tmp,
            BigDecimal.ROUND_HALF_UP));
        break;

    default:
        throw new IllegalStateException("Unknown
            OPERATOR popped");
    }

}

else {
    throw new IllegalArgumentException("Syntax
        error in input");
}

return stack.pop();
}
}

```

Output:

The result is 6838458446744073709551614.466860

Multidimensional Arrays in Java

- Java arrays can hold any reference type. Because an array is a reference type, it follows that you can have arrays of arrays or, in other terminology, *multidimensional* arrays.
- **Multidimensional Arrays** can be defined in simple words as array of arrays.
- Each array has its own length attribute,
 - the columns of a two-dimensional array, do not all have to be the same length (see [Figure 7-3](#)).

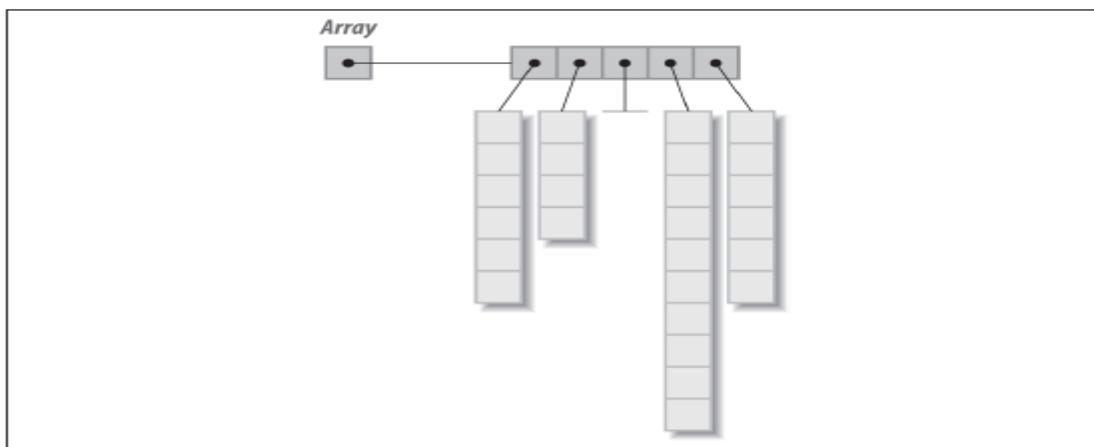


Figure 7-3. Multidimensional arrays

Example: Simple code to allocate a couple of two-dimensional arrays, one using a loop and the other using an initializer. Both are selectively printed.

```
public class MultiArrayDemo {  
    /** Return list of subscript names (unrealistic; just for demo). */  
    public static String[][] getArrayInfo() {  
        String info[][];  
        info = new String[10][10];  
  
        for (int i=0; i<info.length; i++) {  
            for (int j = 0; j<info[i].length; j++) {  
                info[i][j] = "String[" + i + "," + j + "]";  
            }  
        }  
        return info;  
    }  
}
```

```

}

/** Return list of allowable parameters (Applet method). */
public static String[][] getParameterInfo() {
    String param_info[][] = {
        {"fontsize", "9-18", "Size of font"},
        {"URL", "-", "Where to download"},
    };
    return param_info;
}

/** Run both initialization methods and print part of the results */
public static void main(String[] args) {
    print("from getArrayInfo", getArrayInfo());
    print("from getParameterInfo", getParameterInfo());
}

/** Print selected elements from the 2D array */
public static void print(String tag, String[][] array) {

    System.out.println("Number of rows: " + array.length);
    System.out.println("Number of columns: " + array[0].length);
    System.out.println("Array " + tag + " is " + array.length + " x " +
                       array[0].length);

    System.out.println("Array[0][0] = " + array[0][0]);
    System.out.println("Array[0][1] = " + array[0][1]);
    System.out.println("Array[0][2] = " + array[0][2]);
    System.out.println("Array[1][0] = " + array[1][0]);
    System.out.println("Array[1][1] = " + array[1][1]);
    System.out.println("Array[1][2] = " + array[1][2]);
}
}

```

Output:

Number of rows: 10
 Number of columns: 10
 Array from getArrayInfo is 10 x 10
 Array[0][0] = String[0,0]
 Array[0][1] = String[0,1]
 Array[0][2] = String[0,2]
 Array[1][0] = String[1,0]

```
Array[1][1] = String[1,1]
Array[1][2] = String[1,2]
Number of rows: 2
Number of columns: 3
Array from getParameterInfo is 2 x 3
Array[0][0] = fontsize
Array[0][1] = 9-18
Array[0][2] = Size of font
Array[1][0] = URL
Array[1][1] = -
Array[1][2] = Where to download
```

Multidimensional Collections in Java

- Multidimensional Collections (or Nested Collections) is a collection of group of objects where each group can have any number of objects dynamically. Hence, here we can store any number of elements in a group whenever we want.
- Syntax for One-dimensional arraylist
 - `ArrayList <Object> x= new ArrayList <Object>();`
- Syntax for Multi-dimensional arraylist
 - `ArrayList<ArrayList<Object>> a = new ArrayList<ArrayList<Object>>();`
- Example of Multidimensional ArrayList:
 - `[[3, 4], [12, 13, 14, 15], [22, 23, 24], [33]]`
- Add Method for Multidimensional ArrayList in Java:
 - `boolean add(ArrayList<Object> e)`
 - It is used to insert elements in the specified collection.
 - `void add(int index, ArrayList<Object> e)`
 - It is used to insert the elements at specified position in a Collection.

Example: Java program for Multidimensional ArrayList.

```
import java.util.*;

class MultidimensionalArrayList {

    /*function for creating and returning 2D ArrayList*/
    static List create2DArrayList() {

        /*Declaring 2D ArrayList*/
        ArrayList<ArrayList<Integer>> x =
            new ArrayList<ArrayList<Integer>>();

        /*one space allocated for 0th row*/
        x.add(new ArrayList<Integer>());

        /*Adding 3 to 0th row created above x(0, 0)*/
        x.get(0).add(0, 3);

        /*Creating 1st row and adding values
        (another way for adding values in 2D collections)*/
        x.add(new ArrayList<Integer>(Arrays.asList(3, 4, 6)));

        /*Add 366 to 1st row 0th column x(1, 0)*/
        x.get(1).add(0, 366);

        /*Add 576 to 1st row 4th column x(1, 4)*/
        x.get(1).add(4, 576);

        /*Adding values to 2nd row*/
        x.add(2, new ArrayList<>(Arrays.asList(3, 84)));

        /*Adding values to 3rd row*/
        x.add(new ArrayList<Integer>(Arrays.asList(83, 6684, 776)));

        /*Adding values to 4th row*/
        x.add(new ArrayList<>(Arrays.asList(8)));

        return x;
    }
}
```

```

public static void main(String args[]) {
    System.out.println("2D ArrayList :");
    /*Printing 2D ArrayList*/
    System.out.println(create2DArrayList());
}
}

```

Output:

2D ArrayList :
[[3], [366, 3, 4, 6, 576], [3, 84], [83, 6684, 776], [8]]

Example: Java program for Multidimensional LinkedHashSet

```

import java.util.*;
class MultidimensionalLinkedHashSet{
    /*function for creating and returning 2D LinkedHashSet*/
    static Set create2DLinkedHashSet() {
        /*Declaring 2D LinkedHashSet*/
        LinkedHashSet<LinkedHashSet<String>> x =
            new LinkedHashSet<LinkedHashSet<String>>();
        /*Creating 1st row */
        x.add(new LinkedHashSet<String>(Arrays.asList("Apple", "Orange")));
        // Creating 2nd row, here "Coffee" will be considered
        // as only one object to maintain uniqueness
        x.add(new LinkedHashSet<String>(Arrays.asList("Tea", "Coffee",
            "Milk", "Coffee", "Water")));
        /* Creating 3rd row */
        x.add(new LinkedHashSet<String>( Arrays.asList("Tomato", "Potato",
            "Onion")));
        // Creating 4th row but it will not be added as it contains the
        // same items as 3rd row (LinkedHashSet inserts only unique items)
        x.add(new LinkedHashSet<String>(Arrays.asList("Tomato", "Potato",
            "Onion")));
    return x;
}

```

```
public static void main(String[] args) {  
    System.out.println("2D LinkedHashSet :");  
  
    /*Printing 2D LinkedHashSet*/  
    System.out.println(create2DLinkedHashSet());  
}  
}
```

Output:

2D LinkedHashSet :

[[Apple, Orange], [Tea, Coffee, Milk, Water], [Tomato, Potato, Onion]]

Object-Oriented Techniques

- Java is an object-oriented (OO) language in the tradition of Simula-67, SmallTalk, and C++.
- It borrows syntax from C++ and ideas from SmallTalk.
- The Java API has been designed and built on the OO model.

Formatting Objects for Printing with `toString()`

- If you print any object, java compiler internally invokes the `toString()` method on the object. Java “knows” that every object has a `toString()` method because `java.lang.Object` has one and all classes are ultimately subclasses of `Object`.
- The default implementation, in `java.lang.Object`, is just prints the class name, an @sign, and the object’s `hashCode()` value.

Example:

```
public class ToStringWithout {  
  
    int x, y;  
  
    /** Simple constructor */  
    public ToStringWithout(int anX, int aY) {  
        x = anX;  
        y = aY;  
    }  
  
    /** Main just creates and prints an object */  
    public static void main(String[] args) {  
  
        ToStringWithout S=new ToStringWithout(42, 86);  
  
        System.out.println(S);  
    }  
}
```

}

Output:

ToStringWithout@4617c264

toString()

- If you want to represent any object as a string, **toString() method** comes into existence.
- The `toString()` method returns the string representation of the object.
- The `toString()` method inherited from `java.lang.Object`.
- So overriding the `toString()` method, returns the desired output.

Example:

```
public class ToStringWith{
    int x, y;

    /** Simple constructor */
    public ToStringWith(int anX, int aY) {
        x = anX;
        y = aY;
    }

    //override the toString()
    public String toString() {
        return "ToStringWith [" + x + "," + y + "]";
    }

    /** Main just creates and prints an object */
    public static void main(String[] args) {

        ToStringWith S=new ToStringWith (42, 86);

        System.out.println(S);
    }
}
```

Output:

ToStringWith [42,86]

Overriding the equals() and hashCode() Methods

- Java.lang.Object has two very important methods :
 - public boolean equals(Object obj) and
 - public int hashCode().

equals() method

- In java equals() method is used to compare equality of two Objects.
- Simply checks if two Object references (say x and y) refer to the same Object. i.e. It checks if $x == y$.
- Some principles of equals() method of Object class :
 - **reflexive**
 - $x.equals(x)$ must be true.
 - **symmetrical**
 - $x.equals(y)$ must be true if and only if $y.equals(x)$ is also true.
 - **transitive**
 - If $x.equals(y)$ is true and $y.equals(z)$ is true, then $x.equals(z)$ must also be true.
 - **repeatable**
 - Multiple calls on $x.equals(y)$ return the same value (unless state values used in the comparison are changed, as by calling a set method).

- cautious
 - `x.equals(null)` must return `false` rather than accidentally throwing a `NullPointerException`.

Example:

```

public class EqualsDemo {

    static int a = 10, b=20;
    int c;

    // Constructor
    EqualsDemo() {
        System.out.println("Addition of 10 and 20 : ");
        c = a + b;
        System.out.println("Answer : "+ c);
    }

    // Driver code
    public static void main(String args[]) {
        System.out.println("1st object created...");
        EqualsDemo obj1 = new EqualsDemo();
        System.out.println("2nd object created...");
        EqualsDemo obj2 = new EqualsDemo();
        EqualsDemo obj3 = obj1;
        System.out.println("obj1 == obj2 : " + obj1.equals(obj2));
        System.out.println("obj1 == obj3 : " + obj1.equals(obj3));
    }
}

```

Output:

1st object created...
 Addition of 10 and 20 :
 Answer : 30
 2nd object created...
 Addition of 10 and 20 :
 Answer : 30
 obj1 == obj2 : false
 obj1 == obj3 : true

hashCode() method

- The hashCode() method is supposed to return an int that should uniquely identify different objects.
- Hashcode value is mostly used in hashing based collections like HashMap, HashSet, HashTable....etc.
- A properly written hashCode() method will follow these rules:
 - It is repeatable.
 - hashCode(x) must return the same int when called repeatedly, unless set methods have been called.
 - It is consistent with equality.
 - If x.equals(y), then x.hashCode() must == y.hashCode().
 - Distinct objects should produce distinct hashCodes
 - If !x.equals(y), it is not required that x.hashCode() != y.hashCode(), but doing so may improve performance of hash tables (i.e., hashes may call hashCode() before equals()).

Example:

```
class SomeClass{  
}  
  
public class PrintHashCodes {  
  
    /** Some objects to hashCode() on */  
    protected static Object[] data = {  
        new PrintHashCodes(),  
        new java.awt.Color(0x44, 0x88, 0xcc),  
        new SomeClass()  
    };  
}
```

```

public static void main(String[] args) {

    System.out.println("About to hashCode " + data.length + " objects.");
    for (int i=0; i<data.length; i++) {
        System.out.println(data[i].toString() + " --> " + data[i].hashCode());
    }
    System.out.println("All done.");
}
}

```

Output:

About to hashCode 3 objects.
PrintHashCodes@27d6c5e0 --> 668386784
java.awt.Color[r=68,g=136,b=204] --> -12285748
SomeClass@12edcd21 --> 317574433
All done.

Example: Java program to illustrate how hashCode() and equals() methods work.

```

import java.io.*;

class Geek {

    public String name;
    public int id;

    Geek(String name, int id) {

        this.name = name;
        this.id = id;
    }
}

```

```
@Override
public boolean equals(Object obj) {

    // checking if both the object references are
    // referring to the same object.
    if(this == obj)
        return true;

    // it checks if the argument is of the
    // type Geek by comparing the classes
    // of the passed argument and this object.
    // if(!(obj instanceof Geek)) return false; ---> avoid.
    if(obj == null || obj.getClass() != this.getClass())
        return false;

    // type casting of the argument.
    Geek geek = (Geek) obj;

    // comparing the state of argument with
    // the state of 'this' Object.
    return (geek.name == this.name && geek.id == this.id);
}
```

```
@Override
public int hashCode()
{

    // We are returning the Geek_id
    // as a hashcode value.
    // we can also return some
    // other calculated value or may
    // be memory address of the
    // Object on which it is invoked.
    // it depends on how you implement
    // hashCode() method.
    return this.id;
}
```

}

```

//Driver code
public class HashCodeEqualsTo {

    public static void main (String[] args) {

        // creating the Objects of Geek class.
        Geek g1 = new Geek("aa", 1);
        Geek g2 = new Geek("aa", 1);

        // comparing above created Objects.
        if(g1.hashCode() == g2.hashCode())
        {
            if(g1.equals(g2))
                System.out.println("Both Objects are equal. ");
            else
                System.out.println("Both Objects are not equal. ");
        }
        else
            System.out.println("Both Objects are not equal. ");
    }
}

```

Output:

Both Objects are equal.

Using Inner Classes

- Java inner class or nested class (a class within a class).
- Java inner class is a class which is declared inside the class or interface.
- The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

- There are basically four types of inner classes in Java.
 - Nested Inner class
 - Static nested classes
 - Method Local inner classes
 - Anonymous inner classes

1) Nested Inner class

- To access the inner class, create an object of the outer class, and then create an object of the inner class.

Example:

```

class OuterClass {
    int x = 10;

    class InnerClass {
        int y = 5;

        //    //If you don't want outside objects to access
        //    //the inner class, declare the class as private
        //    private class InnerClass {
        //        int y = 5;
        //    }
    }

    public class InnerClassExample {

        public static void main(String[] args) {

            //create an object of the outer class
            OuterClass myOuter = new OuterClass();
        }
    }
}
```

```

//create an object of the inner class by using
//an object of the outer class
OuterClass.InnerClass myInner = myOuter.new InnerClass();

System.out.println(myInner.y + myOuter.x);
}
}

```

Output:

15

2) Static nested classes

- An inner class can also be static, which means that you can access it without creating an object of the outer class.

Example:

```

class OuterClass {
    int x = 10;

    static class InnerClass {
        int y = 5;
    }
}

public class InnerClassExample {

    public static void main(String[] args) {

        //create an object of the inner class without creating
        // an object of the outer class
        OuterClass.InnerClass myInner = new OuterClass.InnerClass();
        System.out.println(myInner.y);
    }
}

```

Output:

3) Method Local inner classes

- In Java, we can write a class within a method and this will be a local type. Like local variables, the scope of the inner class is restricted within the method.
- A method-local inner class can be instantiated only within the method where the inner class is defined.

Example:

```
public class Outerclass {
    //instance method of the outer class
    void outermethod() {
        int num = 23;

        //inner class define inside the method of outer class.
        class InnerClass {
            //instance method of the inner class
            public void innermethod() {
                System.out.println("This is method inner class " + num);
            }
        } // end of inner class

        // Accessing the inner class
        InnerClass inner = new InnerClass();
        inner.innermethod();
    }

    public static void main(String args[]) {
        // Accessing the outer class
        Outerclass outer = new Outerclass();
        outer.outermethod();
    }
}
```

}

Output:

This is method inner class 23

4) Anonymous inner classes

- An inner class declared without a class name is known as an **anonymous inner class**.
- In case of anonymous inner classes, we declare and instantiate them at the same time.
- Generally, they are used whenever you need to override the method of a class or an interface.
- The syntax of an anonymous inner class is as follows –

```
AnonymousInner an_inner = new AnonymousInner(){  
    public void my_method() {  
        .....  
        .....  
    }  
};
```

- The anonymous inner class is created in two ways.
 - *As subclass of specified type*
 - *As implementer of the specified interface*

Example: The anonymous inner class is created *as a subclass of specified type*

```
class Demo {  
    void show() {  
        System.out.println("i am in show method of super class");  
    }  
}  
  
public class InnerClassExample {  
  
    // An anonymous class with Demo as base class  
    static Demo d = new Demo() {  
  
        //method of anonymous class  
        void show() {  
  
            //method of Demo class  
            super.show();  
            System.out.println("i am in anonymous class");  
        }  
    };  
  
    public static void main(String[] args){  
        d.show();  
    }  
}
```

Output:

i am in show method of super class
i am in anonymous class

Note: In the above code, we have two class Demo and InnerClassExample. Here demo act as super class and anonymous class acts as a subclass, both classes have a method show(). In anonymous class show() method is overridden.

Example: The anonymous inner class is created *as implementer of the specified interface*

```
public class InnerClassExample {  
  
    // An anonymous class that implements Hello interface  
    static Hello h = new Hello() {  
        public void show() {  
            System.out.println("i am in anonymous class");  
        }  
    };  
  
    public static void main(String[] args) {  
        h.show();  
    }  
  
    interface Hello {  
        void show();  
    }  
}
```

Output:

i am in anonymous class

Note: In the above code we create an object of anonymous inner class, but this anonymous inner class is an implementer of the interface Hello. Any anonymous inner class can implement only one interface at one time. It can either extend a class or implement an interface at a time.

Polymorphism/Abstract Methods

Problem

- You want each of a number of subclasses to provide its own version of one or more methods.

Solution

- Make the method abstract in the parent class; this makes the compiler ensure that each subclass implements it.

Abstract class in Java

- A class which is declared as abstract is known as an abstract class.
- It can have abstract and non-abstract methods.
- It needs to be extended and its method implemented.
- It cannot be instantiated.

Abstract Method in Java

- A method which is declared as abstract and does not have implementation is known as an abstract method.

Polymorphism in Java

- Polymorphism is the ability of an object to take on many forms.
- Java makes software more reliable and maintainable with the use of polymorphism.
- Polymorphism is a great boon for software maintenance: if a new subclass is added, the code in the main program does not change.

Example:

```
//parent class
public abstract class Shape {
    protected int x, y;
    public abstract double computeArea();
}
```

```
public class Rectangle extends Shape {  
    double width=5, height=7;  
    public double computeArea() {  
        return width * height;  
    }  
}  
  
//  
public class Circle extends Shape {  
    double radius=5;  
    public double computeArea() {  
        return Math.PI * radius * radius;  
    }  
}  
  
public class Triangle extends Shape {  
    double a=3.0, b=4, c=5;  
    public double computeArea() {  
        double s = (a + b + c) / 2;  
        return Math.sqrt(s * (s - a) * (s - b) * (s - c));  
    }  
}  
  
import java.util.ArrayList;  
import java.util.Collection;  
  
public class PolymorphismAbstract {  
  
    public static void main(String[] args) {  
  
        // created in a Constructor, not shown  
        Collection<Shape> allShapes;  
        allShapes = new ArrayList<>();  
  
        ArrayList<String> Arr = ArrayList<String>()  
  
        allShapes.add(new Circle());  
        allShapes.add(new Rectangle());  
        allShapes.add(new Triangle());
```

```

// Iterate over all the Shapes, getting their areas;
double totalAreas = 0.0;
int i = 1;

for (Shape s : allShapes) {
    System.out.println("The " + i + " Area : " + s.computeArea());
    totalAreas += s.computeArea();
    i++;
}

System.out.println("Total Area = " + totalAreas);
}
}

```

Output:

The 1 Area: 78.53981633974483
 The 2 Area: 35.0
 The 3 Area: 6.0
 Total Area = 119.53981633974483

Passing Values

Problem

- You need to pass a number like an int into a routine and get back the routine's updated version of that value in addition to the routine's return value.

Solution

- Use a specialized class such as the MutableInteger class presented here.
- MutableInteger, that is like an Integer but specialized by omitting the overhead of Number and providing only the set, get, and incr operations.

- The latter is overloaded to provide a no-argument version that performs the increment (++) operator on its value, and also a one-integer version that adds that increment into the value (analogous to the += operator).
- Because Java doesn't support operator overloading, the calling class has to call these methods instead of invoking the operations syntactically, as you would on an int.

Example:

```

public class StringParse {

    /** This is the function that has a return value of true but
     * also "passes back" the offset into the String where a
     * value was found. Contrived example!
    */

    public static boolean parse(String in, char lookFor,
                                MutableInteger whereFound) {

        int i = in.indexOf(lookFor);

        if (i == -1)
            return false; // not found
        whereFound.setValue(i); // say where found
        return true; // say that it was found
    }

    public static void main(String[] args) {

        // Create object of MutableInteger class
        MutableInteger mi = new MutableInteger();
        String text = "Hello, World";
        char c = 'W';
        if (parse(text, c, mi)) {
            System.out.println("Character " + c + " found at offset "
                               + mi + " in " + text);
        }

        else {
    }
}

```

```
        System.out.println("Not found");
    }
}
}

/** A MutableInteger is like an Integer but mutable, to avoid the
 * excess object creation involved in
 * c = new Integer(c.getInt()+1)
 * which can get expensive if done a lot.
 * Not subclassed from Integer, since Integer is final (for performance :-))
 */
public class MutableInteger {

    private int value = 0;

    public MutableInteger(int i) {
        value = i;
    }

    public MutableInteger() {
        this(0);
    }

    public int incr() {
        value++;
        return value;
    }

    public int incr(int amt) {
        value += amt;
        return value;
    }

    public int decr() {
        value--;
        return value;
    }

    public int setValue(int i) {
```

```

        value = i;
        return value;
    }

public int getValue() {
    return value;
}

public String toString() {
    return Integer.toString(value);
}

public static String toString(int val) {
    return Integer.toString(val);
}

public static int parseInt(String str) {
    return Integer.parseInt(str);
}
}

```

Output:

Character W found at offset 7 in Hello, World

Using Typesafe Enumerations

Problem

- You need to manage a small list of discrete values within a program.

Solution

- Use the Java enum mechanism.

Java Enums

- The **Enum in Java** is a data type which contains a fixed set of constants.
- Java Enums can be thought of as classes which have a fixed set of constants (a variable that does not change).
- The Java enum constants are static and final implicitly. It is available since JDK 1.5.
- The **enum** data type (also known as Enumerated Data Type) is used to define an enum in Java.
- Java Enum internally inherits the *Enum class*, so it cannot inherit any other class, but it can implement many interfaces.
- Java enum can have fields, constructors, methods, and main methods.
- Enums can be used in switch statements.

Example: A code sample showing the definition of a typesafe enum. Simply, iterate over the array returned by the enum class's inherited values() method.

```
public class EnumExample {

    //defining the enum inside the class
    public enum Media {
        BOOK, MUSIC_CD, MUSIC_VINYL, MOVIE_VHS,
        MOVIE_DVD;
    }

    //main method
    public static void main(String[] args) {

        //traversing the enum
        for (Media m : Media.values())
            System.out.println(m);
    }
}
```

```
    }  
}
```

Output:

```
BOOK  
MUSIC_CD  
MUSIC_VINYL  
MOVIE_VHS  
MOVIE_DVD
```

- **Note:** The enum can be defined within or outside the class because it is similar to a class. The semicolon (;) at the end of the enum constants are optional.

Example: Simple program of applying Enum on a switch statement.

```
class EnumExample{  
  
    enum Day{SUNDAY, MONDAY, TUESDAY, WEDNESDAY,  
             THURSDAY, FRIDAY, SATURDAY}  
  
    public static void main(String args[]) {  
  
        Day day=Day.MONDAY;  
  
        switch(day) {  
  
            case SUNDAY:  
                System.out.println("Sunday");  
                break;  
  
            case MONDAY:  
                System.out.println("Monday");  
                break;  
  
            case TUESDAY:  
                System.out.println("Tuesday");  
        }  
    }  
}
```

```

break;

case WEDNESDAY:
    System.out.println("Tuesday");
    break;

case THURSDAY:
    System.out.println("Tuesday");
    break;

case FRIDAY:
    System.out.println("Tuesday");
    break;

case SATURDAY:
    System.out.println("Tuesday");
    break;

default:
    System.out.println("other day");
}

}
}

```

Output:

Monday

Example: An example of an enum with method overriding.

```

public enum MediaFancy {

    /** The enum constant for a book, with a method override */
    BOOK {
        public String toString() {
            return "Book";
        }
    },
}
```

```

/** The enum constant for a Music CD */
MUSIC_CD,
/** ... */
MUSIC_VINYL,
MOVIE_VHS,
MOVIE_DVD;

public static void main(String[] args) {

    /** It is generally disparaged to have a main() in an enum;
    MediaFancy[] data = { BOOK, MOVIE_DVD,
                           MUSIC_VINYL };

    for (MediaFancy mf : data) {
        System.out.println(mf);
    }
}
}

```

Output:

Book
 MOVIE_DVD
 MUSIC_VINYL

Enforcing the Singleton Pattern

Problem

- You want to be sure there is only one instance of your class in a given Java Virtual Machine.

Solution

- Make your class enforce the Singleton Pattern

Singleton Pattern

- The Singleton's purpose is to control object creation, limiting the number of objects to only one.
- Since there is only one Singleton instance, any instance fields of a Singleton will occur only once per class, just like static fields.
- Singletons often control access to resources, such as database connections or sockets.
- For example, if you have a license for only one connection for your database or your JDBC driver has trouble with multithreading, the Singleton makes sure that only one connection is made or that only one thread can access the connection at a time.

Example:

```
public class Singleton {  
  
    private static Singleton singleton = new Singleton();  
  
    /* A private Constructor prevents any other  
     * class from instantiating.  
     */  
    private Singleton() {}  
  
    //Static 'instance (i.e., getInstance( ))' method  
    //which must be public) then simply returns this instance  
    public static Singleton getInstance() {  
        return singleton;  
    }  
  
    /* Other methods protected by singleton-ness */  
    protected static void demoMethod() {  
        System.out.println("demoMethod for singleton");  
    }  
}
```

```
public class SingletonDemo {  
  
    public static void main(String[] args) {  
  
        // create a singleton object  
        Singleton tmp = Singleton.getInstance( );  
        tmp.demoMethod( );  
    }  
}
```

Output:

demoMethod for singleton

Roll Your Own Exceptions

- You can create your own exceptions in Java.
- need to extend the predefined **Exception** class to create your own Exception, which are considered to be checked exceptions.
- When subclassing either of these, it is customary to provide at least these constructors:
 - A no-argument constructor
 - A one-string argument constructor
 - A two argument constructor—a string message and a Throwable “cause”

Example: Exception using in banking problem

- In this example we have used 3 classes:
 - InsufficientFundsException Class:
 - The InsufficientFundsException class is a user-defined exception that extends the Exception class, making it a checked exception. An exception class is like any other class, containing useful fields and methods.
 - CheckingAccount Class:
 - The CheckingAccount class contains a withdraw() method that throws an InsufficientFundsException.
 - BankDemo Class:
 - The BankDemo program demonstrates invoking the deposit() and withdraw() methods of CheckingAccount.

```
import java.io.*;
```

```
public class InsufficientFundsException extends Exception {
```

```
    private double amount;
```

```
    public InsufficientFundsException(double amount) {
```

```
        this.amount = amount;
```

```
    }
```

```
    public double getAmount() {
```

```
        return amount;
```

```
    }
```

```
}
```

```
import java.io.*;
```

```
public class CheckingAccount {
```

```
    private double balance;
```

```
private int number;

public CheckingAccount(int number) {
    this.number = number;
}

public void deposit(double amount) {
    balance += amount;
}

public void withdraw(double amount) throws
    InsufficientFundsException {
    if(amount <= balance) {
        balance -= amount;
    }
    else {
        double needs = amount - balance;
        throw new InsufficientFundsException(needs);
    }
}

public double getBalance() {
    return balance;
}

public int getNumber() {
    return number;
}

public class BankDemo {

    public static void main(String [] args) {

        CheckingAccount c = new CheckingAccount(101);
        System.out.println("Depositing $500...");
        c.deposit(500.00);
        System.out.println("The Curreent Balance: " + c.getBalance());
    }
}
```

```
try {
    System.out.println("\nWithdrawing $100...");
    c.withdraw(100.00);
    System.out.println("The Current Balance: " + c.getBalance());

    System.out.println("\nWithdrawing $600...");
    c.withdraw(600.00);
}
catch (InsufficientFundsException e) {

    System.out.println("Sorry, but you are short $" + e.getAmount());
    e.printStackTrace();
}
}
```

Output:

Depositing \$500...
The Current Balance: 500.0

Withdrawing \$100...
The Current Balance: 400.0

Withdrawing \$600...
Sorry, but you are short \$200.0
InsufficientFundsException
at CheckingAccount.withdraw([CheckingAccount.java:22](#))
at BankDemo.main([BankDemo.java:16](#))

Input and Output

- Java brings various Streams with its I/O package that helps the user to perform all the input-output operations.
- These streams support all the types of objects, data-types, characters, files etc to fully execute the I/O operations.



Standard or Default Streams

- Java provides 3 most common standard or default streams, that are:
 - System.in
 - System.out
 - System.err
- **System.in:** This is the standard input stream that is used to read characters from the keyboard or any other standard input device.
- **System.out:** This is the standard output stream that is used to produce the result of a program on an output device like the computer screen.
 - Here is a list of the various print functions that we use to output statements:
 - **print():** This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console

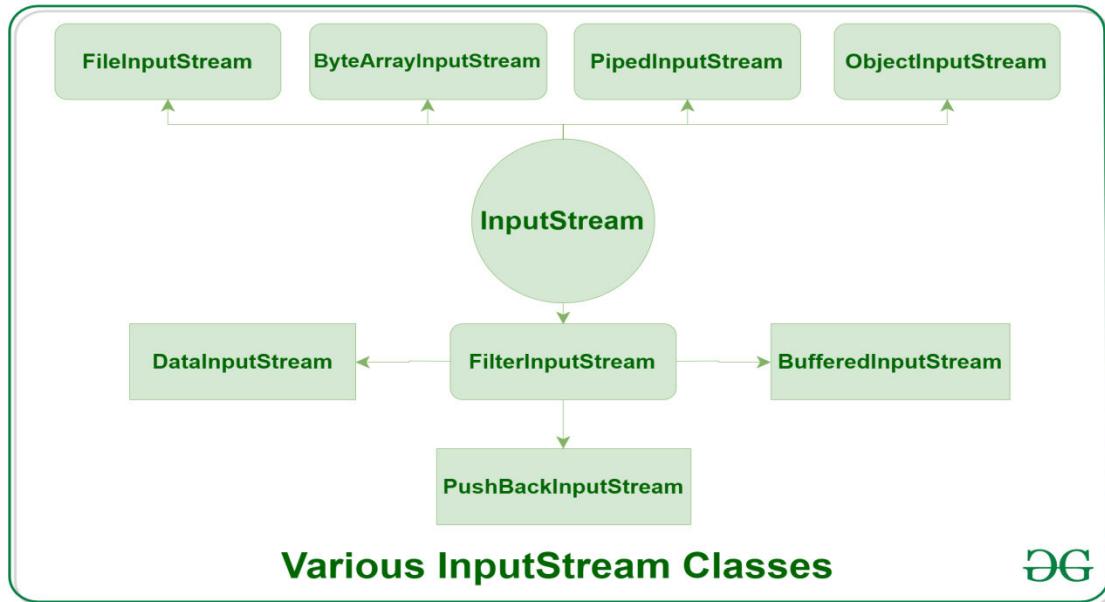
and the cursor remains at the end of the text at the console. The next printing takes place from just here.

- **println():** This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.
- **printf():** This is the easiest of all methods as this is similar to printf in C.
 - Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.
- **System.err:** This is the standard error stream that is used to output all the error data that a program might throw, on a computer screen or any standard output device. This stream also uses all the 3 above-mentioned functions to output the error data:
 - print()
 - println()
 - printf()

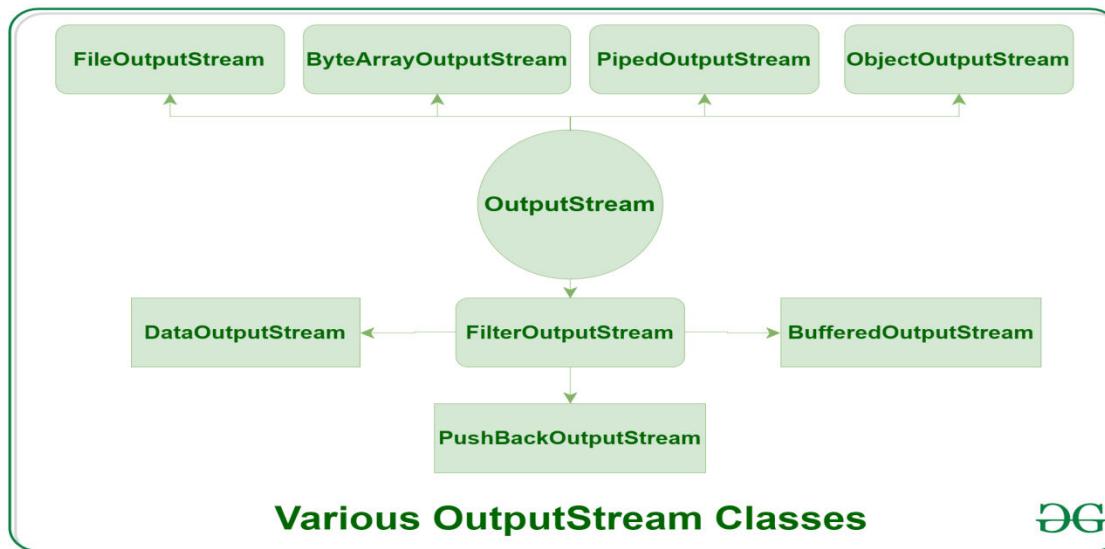
Types of Streams:

- The types of stream broadly divided into 2 types
 - Depending on the type of operations
 - Depending on the types of file
- **Depending on the type of operations**, the streams can be divided into two primary classes:
 - Input Stream
 - Output Stream

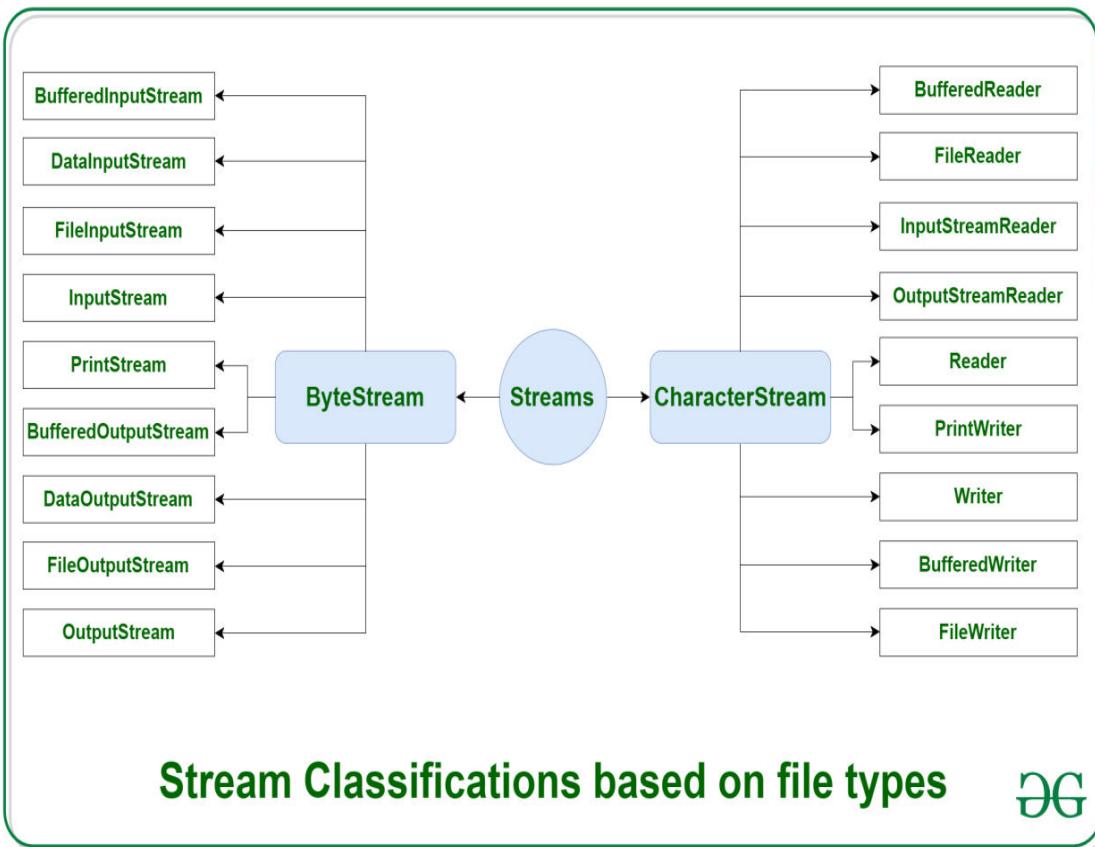
- **Input Stream:** These streams are used to read data that must be taken as an input from a source array or file or any peripheral device. For e.g., FileInputStream, BufferedInputStream, ByteArrayInputStream etc.



- **Output Stream:** These streams are used to write data as outputs into an array or file or any output peripheral device. For e.g., FileOutputStream, BufferedOutputStream, ByteArrayOutputStream etc.



- **Depending on the types of file**, Streams can be divided into two primary classes which can be further divided into other classes as can be seen through the diagram below, followed by the explanations.
 - **ByteStream**
 - **CharacterStream**



- **ByteStream:** This is used to process a data byte by byte (8 bits). Though it has many classes, the `FileInputStream` and the `FileOutputStream` are the most popular ones. The `FileInputStream` is used to read from the source and `FileOutputStream` is used to write to the destination. Here are the list of various ByteStream Classes:

Stream class	Description
<u>BufferedInputStream</u>	It is used for Buffered Input Stream.
<u>DataInputStream</u>	It contains a method for reading java standard datatypes.
<u>FileInputStream</u>	This is used to read from a file
<u>InputStream</u>	This is an abstract class that describes stream input.
<u>PrintStream</u>	This contains the most used print() and println() method
<u>BufferedOutputStream</u>	This is used for Buffered Output Stream.
<u>DataOutputStream</u>	This contains method for writing java standard data types.
<u>FileOutputStream</u>	This is used to write to a file.
<u>OutputStream</u>	This is an abstract class that describes stream output.

- **CharacterStream:** In Java, the characters are stored using Unicode conventions. Character stream automatically allows us to read/write data character by character. Though it has many classes, the FileReader and the FileWriter are the most popular ones. FileReader and FileWriter are character streams used to read from the source and write to the destination respectively. Here are the list of various CharacterStream Classes:

Stream class	Description
<u>BufferedReader</u>	It is used to handle buffered input stream.
<u>FileReader</u>	This is an input stream that reads from file.
<u>InputStreamReader</u>	This input stream is used to translate byte to character.
<u>OutputStreamWriter</u>	This output stream is used to translate character to byte.
<u>Reader</u>	This is an abstract class that define character stream input.
<u>PrintWriter</u>	This contains the most used print() and println() method
<u>Writer</u>	This is an abstract class that define character stream output.
<u>BufferedWriter</u>	This is used to handle buffered output stream.
<u>FileWriter</u>	This is used to output stream that writes to file.

Reading Standard Input

In Java, there are three different ways of reading input from the user in the command line environment(console).

- Using BufferedReader Class
- Using Scanner Class
- Using Console Class

Using Buffered Reader Class

- This is the Java classical method to take input, Introduced in JDK1.0.
- We can read input from the user in the command line.
- This method is used by wrapping the System.in (standard input stream) in an InputStreamReader which is wrapped in a BufferedReader.
- InputStreamReader class can be used to read data from the keyboard. It performs two tasks:
 - Connects to the input stream of keyboard.
 - Converts the byte-oriented stream into character-oriented stream.
- BufferedReader class can be used to read the data line by line by readLine() method.
- **Advantages:** The input buffers for efficient reading.
- **Drawback:** The wrapping code is hard to remember.

Example: Java program to demonstrate BufferedReader.

```
import java.io.*;
public class BufferReaderEX {

    public static void main(String args[])throws Exception{

        //reading input
        BufferedReader br = new BufferedReader(new
                                         InputStreamReader(System.in));

        //alternate method for reading input
        InputStreamReader r=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(r);

        System.out.println("Enter your name : ");
        String name=br.readLine();
    }
}
```

```
        System.out.println("Welcome " + name);
    }
}
```

Output:

Enter your name :

ABC

Welcome ABC

Example: Program to read an Integer from the standard input.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class ReadStdinInt {
    public static void main(String[] ap) {
        String line = null;
        int val = 0;
        try {
            BufferedReader is = new BufferedReader( new
                InputStreamReader(System.in));
            System.out.println("Enter the string");
            line = is.readLine();
            val = Integer.parseInt(line);
            System.out.println("I read this number: " + val);
        } catch (NumberFormatException ne) {
            System.err.println("Not a valid number: " + line);
        } catch (IOException e) {
            System.err.println("Unexpected IO ERROR: " + e);
        }
    }
}
```

Output:

Case 1:

Enter the string

12

I read this number: 12

Case 2:

Enter the string

abc 12

Not a valid number: abc 12

Using Scanner Class

- This is probably the most preferred method to take input.
- The main purpose of the Scanner class is to parse primitive types and strings using regular expressions.
- It is also can be used to read input from the user in the command line.
- The following table explains the methods for reading different types of input data:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads an int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

- **Advantages:** Convenient methods for parsing primitives (`nextInt()`, `nextFloat()`, ...) from the tokenized input and Regular expressions can be used to find tokens.
- **Drawback:** The reading methods are not synchronized

Example: Java program to demonstrate the working of Scanner in Java.

```
import java.util.Scanner;

public class ScannerEx {

    public static void main(String[] args) {

        Scanner myObj = new Scanner(System.in);

        System.out.println("Enter name, age and salary:");

        // String input
        String name = myObj.nextLine();

        // Numerical input
        int age = myObj.nextInt();
        double salary = myObj.nextDouble();

        // Output input by user
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Salary: " + salary);
    }
}
```

Output:

Enter name, age and salary:

ABC

25

5000

Name: ABC

Age: 25

Salary: 5000.0

Using Console Class

- The Java 6 `System.console()` method to obtain a `Console` object, and use its methods.
- It has become a preferred way for reading user's input from the command line.
- It can be used for reading password-like input without echoing the characters entered by the user; the format string syntax can also be used (like `System.out.printf()`).
- **Advantages:**
 - Reading password without echoing the entered characters.
 - Reading methods are synchronized.
 - Format string syntax can be used.
- **Drawback:** Does not work in a non-interactive environment (such as in an IDE).

Example: Java program to demonstrate the working of `System.console()`.

```
public class ConsoleRead {  
  
    public static void main(String[] args)  
    {  
        //Using Console to input data from user  
        String name = System.console().readLine();  
  
        System.out.println(name);  
    }  
}
```

Note that this program does not work on IDEs as `System.console()` may require console.

Example: The Console class is quite useful for reading a password without having it echo.

```
import java.io.Console;

public class ReadPassword {

    public static void main(String[] args) {

        Console cons;

        if ((cons = System.console()) != null) {

            char[] passwd = null;

            try {
                // readPassword() method that takes a prompt
                // argument
                passwd = cons.readPassword("Password: ");
                // In real life you would send the password into
                // authentication code
                System.out.println("Your password was: " + new
                    String(passwd));
            }
            finally {
                // Shred this in-memory copy for security reasons
                if (passwd != null) {
                    java.util.Arrays.fill(passwd, ' ');
                }
            }
        } else {
            throw new RuntimeException("No console, can't get
                password");
        }
    }
}
```

Writing Standard Output or Standard Error

Problem

Suppose, You want your program to write to the standard output or the standard error stream.

Solution

Use System.out or System.err as appropriate.

Example: Java program to illustrate standard System.out or System.err

```
public class DemoPrint {  
    public static void main(String[] args)  
    {  
        System.out.println("Java code to illustrate print() ");  
        // println() print all in  
        // the same line  
        System.out.print("Hello");  
        System.out.print(" "); //print blank space  
        System.out.print("to");  
        System.out.print(" "); //print blank space  
        System.out.print("JAVA");  
  
        //Print the blank line  
        System.out.println(" ");  
  
        int Val = 10;  
        System.out.print("The Value: " + Val);  
  
        //Print the blank line  
        System.out.println(" ");  
  
        System.out.println("Java code to illustrate println() ");  
  
        // println() print all in  
        // the different line  
        System.out.println("Hello");  
        System.out.println("to");  
        System.out.println("JAVA");
```

```

int Data = 20;
System.out.println("The Value: " + Data);

//Java code to illustrate System.err
System.out.println("Java code to illustrate System.err ");
int a=14, b=0;

try {
    int c=a/b;
    System.out.println("The result = " + c);
} catch(Exception e) {
    //else
        //error stream print in red with error
        System.err.println("Catch Error " + e);
}
}
}

```

Output:

Java code to illustrate print()

Hello to JAVA

The Value: 10

Java code to illustrate println()

Hello

to

JAVA

The Value: 20

Java code to illustrate System.err

Catch Error java.lang.ArithmaticException: / by zero

Printing with Formatter and printf

- **Problem**

- You want the ease of use that the java.util.Formatter class brings to simple printing tasks.

- **Solution**

- Use Formatter for printing values with fine-grained control over the formatting.
- **Formatting output using System.out.printf() :**
 - This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments.
- **Formatting output using java.util.Formatter**
 - The **java.util.Formatter** class provides support for layout justification and alignment, common formats for numeric, string, and date/time data, and locale-specific output.
 - The format code consists of a percent sign (%), optionally an argument number followed by a dollar sign (\$), optionally a field width or precision, and a format type (d for decimal integer, that is, an integer with no decimal point, f for floating point, and so on).
 - A simple use might look like the following:

```
format("%1$04d - the year of %2$f", 1956, Math.PI);
```

% - format code
1\$ - use first arg (1951)
0 - leading with 0 if needed
4 - field width (4 digits)
d - decimal integer (int)

% - format code
2\$ - use second arg (PI)
f - floating point

Table : Formatter format codes	
Code	Meaning
c	Character (argument must be char or integral type containing valid character value).
d	“decimal int”—integer to be printed as a decimal (radix 10) with no decimal point (argument must be integral type).

f	Floating-point value with decimal fraction (must be numeric); field width may be followed by decimal point and fractional digit field width; e.g., 7.2f.
e	Floating-point value in scientific notation.
g	Floating-point value, as per f or e, depending on magnitude.
s	General format; if value is null, prints “null,” else if arg implements Formattable, format as per arg.formatTo(); else format as per arg.toString().
t	Date codes; follow with secondary code. Argument must be long, Long, Calendar, or Date.
n	Newline; insert the platform-dependent line ending character.
%	Insert a literal % character.

Table : Formatting codes for dates and times

Code	Meaning
Y	Year (at least four digits).
m	Month as 2-digit (leading zeros) number
B	Locale-specific month name (b for abbreviated)
d	Day of month (2 digits, leading zeros)
e	Day of month (1 or 2 digits)
A	Locale-specific day of week (a for abbreviated)
H or I	Hour in 24-hour (H) or 12-hour (I) format (2 digits, leading zeros)
M	Minute (2 digits)
S	Second (2 digits)
P/p	Locale-specific AM or PM in uppercase (P) or lowercase (p)
R or T	24-hour time combination: %tH:%tM ® or %tH:%tM:%tS (T)
D	Date formatted as “%tm/%td/%ty”

Example: Some examples of using a Formatter

```
import java.util.Formatter;

public class FormatterDemo {

    public static void main(String[] args) {
        Formatter fmtr = new Formatter();
        fmtr.format("Hello, my name is %s and I am %d years old.", "Raj", 25);
        System.out.println(fmtr);
    }
}
```

```

Object result = fmtr.format("%1$04d - the year of %2$f",
                           1956, Math.PI);
System.out.println(result);

// Shorter way using static String.format(), and
// default parameter numbering.
Object stringResult = String.format("%04d - the year of %f",
                                     1956, Math.PI);
System.out.println(stringResult);

// A shorter way using PrintStream/PrintWriter.format, more
in line with other languages. But this way you must provide
the newline delimiter using %n (do NOT use \n as that is
platform-dependent!).
System.out.printf("%04d - the year of %f%n", 1956, Math.PI);

// Format doubles with more control
System.out.printf("PI is approximately %4.4f%n", Math.PI);

// Format doubles with more control
System.out.format("PI is approximately %4.2f%n", Math.PI);
}

}

```

Output:

```

1956 - the year of 3.141593
1956 - the year of 3.141593
1956 - the year of 3.141593
PI is approximately 3.1416
PI is approximately 3.14

```

Example: JAVA program for some date examples

```

import java.util.Calendar;
import java.util.Date;

public class FormatterDates {

```

```

public static void main(String[] args) {

    // Format number as dates e.g., 2014-06-28
    System.out.printf("%4d-%02d-%2d%n", 2014, 6, 28);

    // Format fields directly from a Date object: multiple fields
    // from "1$"
    Date today = Calendar.getInstance().getTime();
    // Might print e.g., July 4, 2015:
    System.out.printf("Today is %1$tB %1$td, %1$tY%n", today);
}

}

```

Output:

2014-06-28
 Today is May 20, 2020

Scanning Input with the Scanner Class

Problem

- You want the ease of use that the `java.util.Scanner` class brings to *simple* reading tasks.

Solution

- Use `Scanner`'s `next()` methods for reading.
- `Scanner` class in Java is found in the `java.util` package. Java provides various ways to read input from the keyboard, the `java.util.Scanner` class is one of them.
- The Java `Scanner` class breaks the input into tokens using a delimiter which is whitespace by default. It provides many methods to read and parse various primitive values.
- The Java `Scanner` class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java. By the help of `Scanner` in Java, we can get input from the user in primitive types such as `int`, `long`, `double`, `byte`, `float`, `short`, etc.

Table: Scanner methods

Returned type	“has” method	“next” method	Comment
String	hasNext()	next()	The next complete token from this scanner
String	hasNext(Pattern)	next(Pattern)	The next string that matches the given regular expression (regex)
String	hasNext(String)	next(String)	The next token that matches the regex pattern constructed from the specified string
BigDecimal	hasNextBigDecimal()	nextBigDecimal()	The next token of the input as a BigDecimal
BigInteger	hasNextBigInteger()	nextBigInteger()	The next token of the input as a BigInteger
boolean	hasNextBoolean()	nextBoolean()	The next token of the input as a boolean
byte	hasNextByte()	nextByte()	The next token of the input as a byte
double	hasNextDouble()	nextDouble()	The next token of the input as a double
float	hasNextFloat()	nextFloat()	The next token of the input as a float
int	hasNextInt()	nextInt()	The next token of the input as an int
String	N/A	nextLine()	Reads up to the end-of-line, including the line ending
long	hasNextLong()	nextLong()	The next token of the input as a long
short	hasNextShort()	nextShort()	The next token of the input as a short

Example: Java program to read data of string types using Scanner class.

```
public class ScannerEx {  
  
    public static void main(String[] args) {  
  
        //Create Scanner object  
        Scanner scan = new Scanner("Hello World!");  
  
        //Print the Strings  
        while (scan.hasNext())  
            System.out.println(scan.next());  
    }  
}
```

Output:

Hello
World!

Example: Java program to read data of various types using Scanner class.

```
public class ScannerEx {  
  
    public static void main(String[] args) {  
  
        String sampleDate = "25 Dec 1988";  
        try (Scanner sDate = new Scanner(sampleDate)) {  
  
            //Scans the next token of the input as an int.  
            int dayOfMonth = sDate.nextInt();  
  
            //next() method finds and returns the next complete token.  
            String month = sDate.next();  
  
            int year = sDate.nextInt();  
            System.out.printf("%d-%s-%02d%n", year, month, dayOfMonth);  
        }  
    }  
}
```

Output:

1988-Dec-25

Opening a File by Name

Problem

- The Java documentation doesn't have methods for opening files. How do I connect a filename on disk with a Reader, Writer, or Stream?

Solution

- Construct a FileReader, FileWriter, FileInputStream, or FileOutputStream.

There are following ways to open a file in Java:

- Java Desktop class
- *Java FileInputStream class*
- *Java FileReader class*
- *Java BufferedReader class*
- *Java Scanner class*
- Java nio package

Java FileInputStream class

- The Java FileInputStream class is used to open and read a file.
- We can open and read a file by using the constructor of the FileInputStream class.
- It accepts a file as an argument.
- It throws **FileNotFoundException** if the file does not exist or file name is a directory.

Example: Java Program to illustrate to read the content from Text File using FileInputStream class.

```
import java.io.File;
import java.io.FileInputStream;

public class OpenFileExample {

    public static void main(String args[]) {

        try {
            //constructor of file class having file as argument
            File file = new File("D:\\CSW - 2\\CSW-1 September
                                2020\\DemoJava\\demofile.txt");

            //opens a connection to an actual file
            FileInputStream fis = new FileInputStream(file);

            System.out.println("FILE CONTENT: ");
            int r=0;

            //read() method reads the next byte of the data from the
            // inputstream and returns int in the range of 0 to 255.
            while((r=fis.read())!=-1) {
                System.out.print((char)r);
            }
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

FILE CONTENT:
JAVA
PYTHON
C
C++

Java FileReader class

- The Java **FileReader** class is also used for opening and reading a file.
- It belongs to a **java.io** package.
- It is a convenience for reading characters of the files.
- It is used for reading raw bytes using the **InputStream** class.
- We use the constructor of the **InputStream** class to open and read a file.
- It accepts a file as an argument.
- It throws **FileNotFoundException** if the file does not exist or file name is a directory.

Example: Java Program to illustrate to read the content from Text File using **FileReader** class.

```
import java.io.File;
import java.io.FileReader;

public class OpenFileExample {

    public static void main(String args[]) {

        try {
            //constructor of File class having file as argument
            File file=new File("D:\\CSW - 2\\CSW-1 September
                                2020\\DemoJava\\demofile.txt");

            //creates a File reader input stream
            FileReader fr=new FileReader(file);

            System.out.println("FILE CONTENT: ");
            int r=0;
```

```

//read() method reads the next byte of the data from the
// inputstream and returns int in the range of 0 to 255.
while((r=fr.read())!= -1) {
    System.out.print((char)r);
}
catch(Exception e) {
    e.printStackTrace();
}
}
}

```

Output:

FILE CONTENT:

JAVA
PYTHON
C
C++

Java BufferedReader class

- The Java BufferedReader class reads text from a character input stream. It belongs to a java.io package.
- We use the constructor of the BufferedReader class to open or read a file.
- It creates a buffering character-input stream that uses a default sized input buffer.
- It accepts a file as an argument.
- It throws **FileNotFoundException** if the file does not exist or file name is a directory.

Example: Java Program to illustrate to read the content from Text File using BufferedReader class.

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;

public class OpenFileExample {

    public static void main(String args[]) {

        try {
            //constructor of File class having file as argument
            File file=new File("D:\\CSW - 2\\CSW-1 September
                                2020\\DemoJava\\demofile.txt");

            //creates a buffer reader input stream
            BufferedReader br=new BufferedReader(new FileReader(file));

            System.out.println("FILE CONTENT: ");
            int r=0;

            //read() method reads the next byte of the data from the
            // inputstream and returns int in the range of 0 to 255.
            while((r=br.read())!=-1) {
                System.out.print((char)r);
            }
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
}
```

Output:

FILE CONTENT:

JAVA

PYTHON

C

C++

Java Scanner class

- The Java Scanner class is also used for opening and reading a file.
- The Scanner class belongs to java.util package. The constructor of Scanner class is used for opening and reading a file.
- It accepts a file as an argument.
- It throws FileNotFoundException if the file does not exist or file name is a directory.

Example: Java Program to illustrate to read the content from Text File using Java Scanner class.

```
import java.io.File;
import java.util.Scanner;

public class OpenFileExample {

    public static void main(String args[]) {

        try {
            //constructor of File class having file as argument
            File file=new File("D:\\CSW - 2\\CSW-1 September
                                2020\\DemoJava\\demofile.txt");

            //file to be scanned
            Scanner sc = new Scanner(file);

            System.out.println("FILE CONTENT: ");

            //The hasNextLine() is used to check if there is another line
            //in the input of this scanner. It returns true if it finds another
            //line, otherwise returns false.
            while (sc.hasNextLine())
                System.out.println(sc.nextLine());
        }
        catch(Exception e) {
```

```
        e.printStackTrace();
    }
}

}
```

Output:

FILE CONTENT:

JAVA PYTHON

C

C++

Write To a File

There are following ways to write a file in Java:

- Java FileWriter Class
- Java FileOutputStream Class
- Java BufferedWriter Class

Java FileWriter Class

- The Java FileWriter class is used to write character-oriented data to a file.
- It is character-oriented class which is used for file handling in java.
- It provides method to write string directly.
- FileWriter class together with its write() method to write some text to the file.
- After writing to the file, close it with the close() method.

Example: Java Program to illustrate to write the content to Text File using Java FileWriter class.

```
import java.io.FileWriter;

public class WriteFileExample {

    public static void main(String[] args) {

        try {
            //output file will be created
            FileWriter fw = new FileWriter("D:\\CSW - 2\\CSW-1
                                         September 2020\\DemoJava\\output.txt");

            //write the content in the file
            fw.write("Welcome to JAVA Programming Language.");
            fw.close(); // close

        } catch(Exception e){
            System.out.println(e);
        }

        System.out.println("Success...");
    }
}
```

Output:

Success...

Note: the content "Welcome to JAVA Programming Language.", has been written in the output.txt file.

Java FileOutputStream Class

- Java FileOutputStream is an output stream used for writing data to a file.
- If you have to write primitive values into a file, use FileOutputStream class.

- You can write byte-oriented as well as character-oriented data through FileOutputStream class.
- But, for character-oriented data, it is preferred to use FileWriter than FileOutputStream.

Example: Java Program to illustrate to write the content (write byte and write string) to Text File using Java FileOutputStream class.

```

import java.io.FileOutputStream;
import java.io.FileWriter;

public class WriteFileExample {

    public static void main(String[] args) {

        try{
            //output file will be created
            FileOutputStream fos = new FileOutputStream("D:\\CSW –
                2\\CSW-1 September 2020\\DemoJava\\output1.txt");

            //Writes the specified byte to this file output stream.
            fos.write(65); // ASCII Code of 'A'

            String s = " Welcome to JAVA Programming Language./";

            //converting string into byte array
            byte b[]=s.getBytes();

            //Writes the specified byte to this file output stream.
            fos.write(b);
            fos.close(); // close

        } catch(Exception e){

            System.out.println(e);
        }
    }
}

```

```
        System.out.println("Success...");  
    }  
}
```

Output:

Success...

Note: the content "A Welcome to JAVA Programming Language." has been written in the output1.txt file.

Java BufferedWriter Class

- The Java BufferedWriter class is used to provide buffering for Writer instances.
- It makes the performance fast.
- It inherits Writer class.
- The buffering characters are used for providing the efficient writing of single arrays, characters, and strings.
- A buffer size needs to be specified, if not it takes Default value.
- An output is immediately set to the underlying character or byte stream by the Writer.

Example: Java Program to illustrate to write the content (write byte and write string) to Text File using BufferedWriter class.

```
import java.io.BufferedWriter;  
import java.io.FileWriter;  
  
public class WriteFileExample {  
  
    public static void main(String[] args) {
```

```

try{
    //initializing FileWriter by creating output file
    FileWriter fw = new FileWriter("D:\\CSW - 2\\CSW-1
        September 2020\\DemoJava\\output2.txt");

    //initializing BufferedWriter
    BufferedWriter bw = new BufferedWriter(fw);

    //Writes the specified byte to this file output stream.
    bw.write(65); // ASCII Code of 'A'

    String s = " Welcome to JAVA Programming Language./";

    //Writes the specified string to the output file.
    bw.write(s);
    bw.close(); // close

} catch(Exception e){

    System.out.println(e);
}
System.out.println("Success...");
}
}

```

Output:

Success...

Note: the content "A Welcome to JAVA Programming Language." has been written in the output2.txt file.

Copying a File

Problem

- You need to copy a file in its entirety.

Solution

- Use a pair of Streams for binary data, or a Reader and a Writer for text, and a while loop to copy until end-of-file is reached on the input.

- To copy the content of one file to another file in java
 - First, we can read the file using FileInputStream and
 - Then write the read content to the output file using FileOutputStream.

Example: Java Program to illustrate to copy the content of one file to another by using FileInputStream class and FileOutputStream class.

```

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

public class CopyExample {

    public static void main(String[] args) {

        FileInputStream instream = null;
        FileOutputStream outstream = null;

        try{
            //Creates a new File instance by converting the
            //given pathname string into an abstract pathname.
            File infile =new File("D:\\CSW - 2\\CSW-1 September
                                2020\\DemoJava\\MyInputFile.txt");

            File outfile =new File("D:\\CSW - 2\\CSW-1 September
                                2020\\DemoJava\\MyOutputFile.txt");

            //Creates a FileInputStream by opening a
            //connection to an actual file
            instream = new FileInputStream(infile);

            //Creates a file output stream to write
            outstream = new FileOutputStream(outfile);

            byte[] buffer = new byte[1024];
    
```

```

int length;

/*copying the contents from input stream to
 * output stream using read and write methods
*/
//Reads up to 1024 bytes of data from this
//input stream into an array of bytes.
while ((length = instream.read(buffer)) > 0){

    //Writes 1024 bytes from the specified byte array
    // starting at offset off to this file output stream.
    outstream.write(buffer, 0, length);
}

//Closing the input/output file streams
instream.close();
outstream.close();

System.out.println("File copied successfully!!");

} catch(IOException ioe) {
    ioe.printStackTrace();
}
}
}
}

```

Output:

File copied successfully!!

Note: the contents “All are Welcomed to JAVA Programming Language.” of MyInputFile.txt have been copied to MyOutputFile.txt.

Reading a File into a String

Problem

- You need to read the entire contents of a file into a string.

Solution

- Use Files.readAllBytes(Paths.get()) method.

Example: Java Program to illustrate reading from text file as string in Java.

```
import java.nio.file.Files;
import java.nio.file.Paths;

public class ReadTextAsString {

    public static String readFileAsString(String fileName) throws Exception{

        String data = "";

        //Files.readAllBytes():Reads all the bytes from a
        //file. The method ensures that the file is closed
        data = new String(Files.readAllBytes(Paths.get(fileName)));
        return data;
    }

    public static void main(String[] args) throws Exception {

        String data = readFileAsString("D:\\CSW - 2\\CSW-1 September
                                         2020\\DemoJava\\MyInputFile.txt");
        System.out.println(data);

    }
}
```

Output:

All are Welcomed to JAVA Programming Language.

Reassigning the Standard Streams

Problem

- You need to reassign one or more of the standard streams System.in, System.out, or System.err.

Solution

- Construct an InputStream or PrintStream as appropriate, and pass it to the appropriate set method in the System class.

- **Standard Input:** This is used to feed the data to the user's program and usually a keyboard is used as a standard input stream and represented as System.in.
- **Standard Output:** This is used to output the data produced by the user's program and usually a computer screen is used for standard output stream and represented as System.out.
- **Standard Error:** This is used to output the error data produced by the user's program and usually a computer screen is used for standard error stream and represented as System.err.

Example: Java Program to illustrate reassign standard streams.

```

import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;

public class ReadTextAsString {

    public static void main(String[] args) {

        try {
            //FileInputStream(): opening a connection to an actual file
            InputStream input = new FileInputStream("c:\\Simple.java");

            //InputStream input = new FileInputStream("D:\\CSW –
            //2\\CSW-1 September 2020\\DemoJava\\MyInputFile.txt");

            System.out.println("File opened... ");

        } catch (IOException e){
            System.err.println("File opening failed:");
        }
    }
}

```

Output:

File opening failed.

Reading/Writing Binary Data

Problem

- You need to read or write binary data, as opposed to text.

Solution

- Use a DataInputStream or DataOutputStream.
- The DataInputStream class reads primitive Java data types from an underlying input stream in a machine-independent way.
- The DataOutputStream class writes primitive Java data types to an output stream in a portable way.

Example: Java program to demonstrate DataInputStream and DataOutputStream.

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;

public class DataStreamExample {

    public static void main(String[] args) throws IOException {

        //Creates a FileInputStream by opening a connection to an actual file
        InputStream input = new FileInputStream("D:\\CSW - 2\\CSW-1
                                                September 2020\\DemoJava\\BinaryInput.txt");

        //Creates a DataInputStream that uses the
        //specified underlying InputStream.
        DataInputStream inst = new DataInputStream(input);

        byte[] arr = new byte[1024];

        //Reads some number of bytes from the input stream
        // and stores them into the buffer array arr.
        inst.read(arr);
```

```

//Creates a file output stream to write to the
//file with the specified name.
FileOutputStream fileOutputStream =
    new FileOutputStream("D:\\CSW - 2\\CSW-1
September 2020\\DemoJava\\BinaryOutput.txt");

//Creates a new data output stream to write data
//to the specified underlying output stream.
DataOutputStream dataOutputStream =
    new DataOutputStream(fileOutputStream);

for (byte bbt : arr) {

    //Writes the specified byte (the low eight bits of
    // the argument b) to the underlying output stream.
    dataOutputStream.write(bbt);
}

//Print the data on computer screen
for (byte bt : arr) {
    char k = (char) bt;
    System.out.print(k);
}
}
}

```

Output:

001
101
110

Seeking to a Position within a File

Problem

- You need to read from or write to a particular location in a file, such as an indexed file.

Solution

- Use a RandomAccessFile.

- **Java.io.RandomAccessFile** class

- This class is used for reading and writing to random access file.
- The RandomAccessFile class also implements the DataInput and DataOutput interfaces.
- A random access file behaves like a large array of bytes.
- There is a cursor implied to the array called file pointer, by moving the cursor we do the read write operations.
- If end-of-file is reached before the desired number of byte has been read than EOFException is thrown. It is a type of IOException.

Example:

```
import java.io.IOException;
import java.io.RandomAccessFile;

public class ReadRandom {

    final static String FILENAME = "D:\\CSW - 2\\CSW-1 September
                                    2020\\DemoJava\\MyFile.txt";

    public static void main(String[] args) {

        try {
            System.out.println(new String(readFromFile(FILENAME, 0, 18)));

            writeToFile(FILENAME, "I love my country and my people", 31);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```

private static byte[] readFromFile(String filePath, int position, int size)
throws IOException {

    //Creates a random access file stream to
    //read from a file with the specified name.
    RandomAccessFile file = new RandomAccessFile(filePath, "r");

    file.seek(position); //Sets the file-pointer offset
    byte[] bytes = new byte[size];
    file.read(bytes);

    file.close();
    return bytes; //Output: This class is used
}

private static void writeToFile(String filePath, String data, int position)
throws IOException {

    //Creates a random access file stream to
    //write to a file with the specified name.
    RandomAccessFile file = new RandomAccessFile(filePath, "rw");
    file.seek(position); //Sets the file-pointer offset

    //getBytes(): Encodes this String into a sequence of bytes using the
    //platform's default charset, storing the result into a new byte array.
    file.write(data.getBytes());
    file.close();
}

```

Output:

This class is used

Before running the above program

The myFile.TXT contains text "This class is used for reading and writing to random access file."

After running the above program

The myFile.TXT contains text " This class is used for reading I love my country and my peoplele."

Directory and Filesystem Operations

- This chapter is largely devoted to `java.io.File` class.
- The `File` class gives you the ability to list directories, obtain file status, rename and delete files on disk, create directories, and perform other filesystem operations.
- Note that many of the methods of this class attempt to modify the permanent file store, or disk file system, of the computer you run them on.

Getting File Information

Problem

- You need to know all you can about a given file on disk.

Solution

- Use a `java.io.File` object.

<i>java.io.File methods</i>		
Modifier and Type	Method	Description
boolean	<code>createNewFile()</code>	It atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
boolean	<code>canWrite()</code>	It tests whether the application can modify the file denoted by this abstract pathname. <code>String[]</code>
boolean	<code>canExecute()</code>	It tests whether the application can execute the file denoted by this abstract pathname.
boolean	<code>canRead()</code>	It tests whether the application can read the file denoted by this abstract pathname.

boolean	isAbsolute()	It tests whether this abstract pathname is absolute.
boolean	isDirectory()	It tests whether the file denoted by this abstract pathname is a directory.
boolean	isFile()	It tests whether the file denoted by this abstract pathname is a normal file.
String	getName()	It returns the name of the file or directory denoted by this abstract pathname.
String	getParent()	It returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
boolean	mkdir()	It creates the directory named by this abstract pathname.
long	length()	This method returns the length of the file denoted by this abstract pathname.
long	lastModified()	This method returns the time that the file denoted by this abstract pathname was last modified.
String	getCanonicalPath()	This method returns the canonical pathname string of this abstract pathname.
boolean	exists()	True if something of that name exists
String[]	list()	Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.
File[]	listFiles()	Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.

Example: Java program to displays the property of a file and directory.

```
import java.io.File;
import java.io.IOException;
import java.util.Date;

public class FileStatus {

    public static void main(String[] argv) throws IOException {
        File f=new File("C:\\\\Users\\\\Sangram\\\\Documents\\\\JavaDemo
                        \\\\MyFile.txt");

        // Print File name
        String n = f.getName();
        System.out.println("File Name: " + n);

        // Print full name
        System.out.println("Canonical Name: " + f.getCanonicalPath());

        // Print parent directory if possible
        String p = f.getParent();
        if (p != null) {
            System.out.println("Parent Directory: " + p);
        }

        // Check if the file is readable
        if (f.canRead()) {
            System.out.println("File is Readable.");
        }

        // Check if the file is writable
        if (f.canWrite()) {
            System.out.println("File is Writable.");
        }
    }
}
```

```
// Report on the modification time.  
Date d = new Date(f.lastModified());  
System.out.println("Last Modified " + d);  
  
// See if file, directory, or other. If file, print size.  
if (f.isFile()) {  
  
    // Report on the file's size  
    System.out.println("File size is " + f.length() + " bytes.");  
  
} else if (f.isDirectory()) {  
  
    System.out.println("It's a Directory");  
  
} else {  
  
    System.out.println("It's Neither a File nor a Directory!");  
  
}  
}  
}
```

Output:

File Name: MyFile.txt
Canonical Name: C:\Users\Sangram\Documents\JavaDemo\MyFile.txt
Parent Directory: C:\Users\Sangram\Documents\JavaDemo
File is Readable.
File is Writable.
Last Modified Thu May 21 22:47:27 IST 2020
File size is 69 bytes.

Creating a File

Problem

- You need to create a new file on disk, but you don't want to write into it.

Solution

- Use a java.io.File object's createNewFile() method.

Example: Java program to create a new file.

```
import java.io.File;
import java.io.IOException;

public class CreateFile {

    public static void main(String[] args) {

        try {
            //creating a Java File instance
            File file = new File("H:\\DemoJava\\javaFile123.txt");

            //createNewFile(): Atomically creates
            //a new, empty file. If file is already
            //exist it show file exist message
            if (file.createNewFile()) {

                System.out.println("New File is created!");

            } else {

                System.out.println("File already exists.");
            }
        }
    }
}
```

```
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
}
```

Output:

New File is created!

Creating a Directory

- You can use the Java File class to create directories if they don't already exists.
- The File class contains the method mkdir().
- The mkdir() returns true if the directory was created, and false if not.

Example: Java program to create a new Directory

```
import java.io.File;  
  
public class CreateDirectory {  
  
    public static void main(String[] args) {  
  
        //Creating a File object  
        File file = new File("H:\\JAVA_DIR");
```

```
//Creating the directory  
boolean bool = file.mkdirs();  
  
if(bool){  
    System.out.println("Directory created successfully");  
}  
else{  
    System.out.println("Sorry couldnt create specified  
                      directory");  
}  
}  
}
```

Output:

Directory created successfully

Renaming a File or Directory

Problem

- You need to change a file's name on disk.

Solution

- Use a java.io.File object's renameTo() method.

Example: Java program to rename a file

```
import java.io.File;  
  
public class Rename {  
  
public static void main(String[] args) {
```

```

//File objects for the existing name
File oldName = new File("H:\\DemoJava\\javaFile123.txt");

//File objects for the new name
File newName = new File("H:\\DemoJava\\JAVA.txt");

//renameTo(): Renames the file denoted
//by this abstract pathname. If file is
//already exist it show Error message
if (oldName.renameTo(newName))

    System.out.println("Renamed successfully");

else

    System.out.println("Error");
}
}

```

Output:

Renamed successfully

Example: Java program to rename a Directory

```

import java.io.File;

public class Rename {

    public static void main(String[] args) {

        //File objects for the existing name
        File oldName = new File("H:\\DemoJava");

```

```

//File objects for the new name
File newName = new File("H:\\Java");

//renameTo(): Renames the Directory denoted
//by this abstract pathname. If Directory is
//already exist it show Error message
if (oldName.renameTo(newName))

    System.out.println("Renamed successfully");

else

    System.out.println("Error");
}
}

```

Output:

Renamed successfully

Deleting a File and Directory

Problem

- You need to delete one or more files from the disk.

Solution

- Use a java.io.File object's delete() method.
- It deletes files (subject to permissions) and directories (subject to permissions and to the directory being empty).

Example: Java program to delete a file

```
import java.io.File;  
  
public class DeleteFile {  
  
    public static void main(String[] args) {  
  
        //File objects for the new name  
        File file = new File("H:\\DemoJava\\JAVA.txt");  
  
        //file.delete(): Deletes the file or  
        //directory denoted by this abstract path name. If  
        //no such file display the failed to delete message  
        if(file.delete())  
        {  
            System.out.println("File deleted successfully");  
        }  
        else  
        {  
            System.out.println("Failed to delete the file");  
        }  
    }  
}
```

Output:

File deleted successfully

Example: Java program to delete a Directory.

```
import java.io.File;

public class DeleteDirectory {

    public static void main(String[] args) {

        //File objects for the existing Directory name
        File DirName = new File("H:\\DemoJava");

        //Function for directory deletion
        deleteDir(DirName);

        System.out.println("The directory is deleted.");
    }

    public static boolean deleteDir(File dir){

        //Returns an array of abstract pathnames
        //denoting the files in the directory
        //denoted by this abstract pathname.
        File[] files = dir.listFiles();

        if(files != null){

            for(File fileName : files){

                if(fileName.isDirectory()){

                    deleteDir(fileName);

                } else {

                    fileName.delete();
                }
            }
        }
    }
}
```

```

        System.err.println("** Deleted " + fileName
                           + " **");
    }
}
return dir.delete();
}
}

```

Output:

** Deleted H:\DemoJava\javaFile1.txt **
** Deleted H:\DemoJava\javaFile2.txt **
** Deleted H:\DemoJava\javaFile3.txt **
The directory is deleted.

Creating a Transient File

Problem

- You need to create a file with a unique temporary filename, or arrange for a file to be deleted when your program is finished.

Solution

- Use a `java.io.File` object's `createTempFile()` or `deleteOnExit()` method.

File `createTempFile()` method

- The `createTempFile()` function creates a temporary file in a given directory (if the directory is not mentioned then a default directory is selected).

- The function generates the filename by using the prefix and suffix passed as the parameters.
- If the suffix is null then the function uses “.tmp” as suffix. The function then returns the created file.
- **Syntax:**
 - *public static File createTempFile(String prefix, String suffix)*
 - **prefix** – The prefix string defines the file name; must be at least three characters long.
 - **suffix** – The suffix string defines the file's extension; if null the suffix ".tmp" will be used.
 - *public static File createTempFile(String prefix, String suffix, File directory)*
 - **prefix** – The prefix string defines the files name; must be at least three characters long.
 - **suffix** – The suffix string defines the file's extension; if null the suffix ".tmp" will be used.
 - **directory** – The directory in which the file is to be created. If the directory is not specified or a null value is passed then the function uses an default directory.

File.deleteOnExit() method

- The ***File.deleteOnExit()*** method also deletes the file or directory defined by abstract pathname.

- The deleteOnExit() method deletes files in reverse order.
- It deletes the file when JVM terminates. It does not return any value.
- Once the request has been made, it is not possible to cancel the request. So this method should be used with care.
- Usually, we use this method when we want to delete the temporary file.
- A temporary file is used to store the less important and temporary data, which should always be deleted when JVM terminates.
- If we want to delete the .temp file manually, we can use File.delete() method.

Example: Java programs illustrate the use of createTempFile() function and deleteOnExit() method.

```

import java.io.File;

public class CreateTempFile {

    public static void main(String[] args) {

        File f = null;

        try {
            System.out.println("*** Create Temporary file using
                            First Syntax ***");

            // creates temporary file
            f = File.createTempFile("tmp", ".txt");
        }
    }
}
```

```
// prints absolute path
System.out.println("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminate
f.deleteOnExit();

// creates temporary file
f = File.createTempFile("tmp", null);

// prints absolute path
System.out.println("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminate
f.deleteOnExit();

System.out.println(" ");

System.out.println("*** Create Temporary file using
Second Syntax ***");

// creates temporary file
f = File.createTempFile("tmp", ".txt", new File("D:/"));

// prints absolute path
System.out.println("File path: "+f.getAbsolutePath());

// deletes file when the virtual machine terminate
f.deleteOnExit();

// creates temporary file
f = File.createTempFile("tmp", null, new File("D:/"));

// prints absolute path
System.out.print("File path: "+f.getAbsolutePath());
```

```

        // deletes file when the virtual machine terminates
        f.deleteOnExit();

    } catch(Exception e) {
        // if any error occurs
        e.printStackTrace();
    }
}

```

Output:

*** Create Temporary file using First Syntax ***
File path: C:\Users\Sangram\AppData\Local\Temp\tmp11964534484036360606.txt
File path: C:\Users\Sangram\AppData\Local\Temp\tmp2999134140143943325.tmp

*** Create Temporary file using Second Syntax ***
File path: D:\tmp6661626881408204835.txt
File path: D:\tmp11537867165325446829.tmp

Changing File Attributes

Problem

- You want to change attributes of a file other than its name.

Solution

- Use `setReadOnly()` or `setLastModified()`.

Java.io.File.setReadOnly() Method

- The `java.io.File.setReadOnly()` method switches the file to read only mode and denies any write operations on the file.
- The `setReadOnly()` method is a part of [File](#) class.

- The `setReadOnly()` function marks the specified file or directory such that only read operations are allowed on the file or directory.
- The function returns **boolean data type**. The function returns true if the File object could be set as Read only else false.

Example: Java programs illustrate the use of `setReadOnly()` method

```
import java.io.File;

public class SetReadOnly {

    public static void main(String[] args) {

        try {

            System.out.println("*** Before File Setting ***");

            //Creates a new File instance
            File file = new
            File("C:\\Users\\Sangram\\Documents\\JavaDemo\\FileOperation.txt");

            //Atomically creates a new, empty file
            file.createNewFile();

            //Get the file name
            String f = file.getName();

            if (file.canRead())

                System.out.println(f + " is Readable");

            else
        }
    }
}
```

```
        System.out.println(f + " is Not Readable");

        if (file.canWrite())

            System.out.println(f + " is Writable");

        else

            System.out.println(f + " is Not Writable");

            file.setReadOnly();

            System.out.println("");


            System.out.println("*** After File Setting ***");

            if (file.canRead())

                System.out.println(f + " is Readable");

            else

                System.out.println(f + " is Not Readable");

                if (file.canWrite())

                    System.out.println(f + " is Writable");

                else

                    System.out.println(f + " is Not Writable");

    } catch(Exception e) {

        e.printStackTrace();
    }
}
```

```
        }  
    }  
}
```

Output:

```
*** Before File Setting ***  
FileOperation.txt is Readable  
FileOperation.txt is Not Writable
```

```
*** After File Setting ***  
FileOperation.txt is Readable  
FileOperation.txt is Not Writable
```

Java.io.File. **setLastModified()**

- The **setLastModified()** method is a part of File class.
- The function sets the last modified time of the file or directory. The function sets the last modified value of the file in milliseconds.
- Syntax:
 - `public boolean setLastModified(long time)`
- This function accepts a long value as parameter which represents the new last modified time.
- The function returns a boolean value which states whether the new last modified time is set or not.

Example: Java programs illustrates the use of `setLastModified()` method

```

// date components
year = 2020;
month = 04;
day = 10;

// date in string
String sDate1 = day+ "/" +month+ "/" +year;

Date date1=new
    SimpleDateFormat("dd/MM/yyyy")
        .parse(sDate1);

// calculate milliseconds
millisec = date1.getTime();

// Check if the last modified time
// can be set to new value
if (f.setLastModified(millisec)) {

    // Display that the last modified time
    // is set as the function returned true
    System.out.println("Last modified time of "
        + FName + " is set");

// last modified time
millisec = f.lastModified();

// calculate date object
dt = new Date(millisec);

// Print on the modification time.
System.out.println(FName + " is (new) Last
    Modified on " + dt);

```

```

        }
    else {

        // Display that the last modified time
        // cannot be set as the function returned false
        System.out.println("Last modified time cannot be set");
    }

} catch(Exception e) {
    // if any error occurs
    e.printStackTrace();
}
}
}
}

```

Output:

```

*** Last Modification Time Before Setting ***
abc.txt is Last Modified on Wed May 27 00:28:04 IST 2020
*** Last Modification Time After Setting ***
Last modified time of abc.txt is set
abc.txt is (new) Last Modified on Fri Apr 10 00:00:00 IST 2020

```

Listing a Directory

Problem

- You need to list the filesystem entries named in a directory.

Solution

- Use a `java.io.File` object's `list()` or `listFiles()` method.

list() method

- The **list()** method is a part of `File` class.
- The `java.io.File.list()` returns the array of files and directories in the directory defined by this abstract path name.
- The method returns null, if the abstract pathname does not denote a directory.
- Syntax:
 - `public String[] list()`
 - This function does not have any parameter
 - `public String[] list(FilenameFilter f)`
 - This function takes `FilenameFilter` object as parameter
 - The function returns a string array, or null value if the file object is file.
 - To list the filesystem entities named in the current directory, just write:
 - `String[] list = new File(".").list()`

Example: Java program to demonstrate the use of `list()` function to find all the files and directories in a given directory and current directory.

```
import java.io.File;
```

```
public class ListingDirectory {
```

```
public static void main(String[] args) {  
  
    // try-catch block to handle exceptions  
    try {  
  
        // Create a file object  
        File f = new File("C:\\Users\\Sangram\\Documents");  
  
        // Get all the names of the files/Directory  
        // present in the given directory  
        String[] DirList = f.list();  
  
        System.out.println("Files/Directories are in the given  
                           directory:");  
  
        // Display the names of the files  
        for (String dir : DirList) {  
            System.out.println(dir);  
        }  
  
        System.out.println();  
  
        // Get list of names from current directory  
        String[] Dirs = new java.io.File(".").list();  
  
        System.out.println("Files/Directories are in the current  
                           directory:");  
  
        // Display the names of the files  
        for (String dirname : Dirs) {  
            System.out.println(dirname);  
        }  
    }  
    catch (Exception e) {
```

```
        System.err.println(e.getMessage());
    }
}
}
```

Output:

Files/Directories are in the given directory:

```
desktop.ini
JavaDemo
My Music
My Pictures
My Videos
```

Files/Directories are in the current directory:

```
.classpath
.project
.settings
bin
src
```

Example: Java program to demonstrate the use of *list(FilenameFilter f)* function to find all the files and directories in a given directory whose names start with “My ”.

```
import java.io.File;
import java.io.FilenameFilter;

public class ListingDirectory {

    public static void main(String[] args) {

        // try-catch block to handle exceptions
```

```

try {

    // Create a file object
    File f = new File("C:\\Users\\Sangram\\Documents");

    // Create a FilenameFilter
    FilenameFilter filter = new FilenameFilter() {

        public boolean accept(File f, String name) {

            return name.startsWith("My");
        }
    };

    // Get all the names of the directory
    // present in the given directory
    // and whose names start with "My"
    String[] DirList = f.list(filter);

    System.out.println("Files/Directories are in the given
                      directory:");

    // Display the names of the files
    for (String dir : DirList) {
        System.out.println(dir);
    }
}

catch (Exception e) {
    System.err.println(e.getMessage());
}
}
}

```

Output:

Files/Directories are in the given directory:

My Music
My Pictures
My Videos

listFiles() method

- The **listFiles()** method is a part of [File](#) class.
- The function returns an array of Files denoting the files in a given abstract pathname if the path name is a directory else returns null.
- Syntax:
 - `public File[] listFiles()`
 - This function does not have any parameter.
 - `public File[] listFiles(FilenameFilter f)`
 - This function takes FilenameFilter object as parameter.
 - `public File[] listFiles(FileFilter f)`
 - This function takes FileFilter object as parameter.
 - The function returns a File array, or null value if the file object is a file.
 - To get an array of already constructed File objects rather than Strings, use:
 - `File[] list = new File(".").listFiles();`

Example: Java program to demonstrate the use of listFiles() method to find all the files and directories in a given directory and current directory.

```
import java.io.File;

public class ListingDirectory {

    public static void main(String[] args) {

        // try-catch block to handle exceptions
        try {

            // Create a file object
            File f = new File("C:\\\\Users\\\\Sangram\\\\Documents");

            // Get all the names of the files/Directory
            // present in the given directory
            File[] DirList = f.listFiles();

            System.out.println("Files/Directories are in the given
                               directory:");

            // Display the names of the files/Directories
            for (int i = 0; i < DirList.length; i++) {
                System.out.println(DirList[i].getName());
            }

            System.out.println();

            // Get list of names from current directory
            File[] Dirs = new File(".").listFiles();

            System.out.println("Files/Directories are in the given
```

```

        directory:");

// Display the names of the files/Directories
for (int j = 0; j < Dirs.length; j++) {
    System.out.println(Dirs[j].getName());
}
}

catch (Exception e) {

    System.err.println(e.getMessage());
}
}
}

```

Output:

Files/Directories are in the given directory:
 desktop.ini
 JavaDemo
 My Music
 My Pictures
 My Videos

Files/Directories are in the given directory:
 .classpath
 .project
 .settings
 bin
 src

Example: Java program to demonstrate the use of `listFiles(FilenameFilter f)` function to find all the files and directories in a given directory whose names start with “out”.

```
import java.io.File;
import java.io.FilenameFilter;

public class ListingDirectory {

    public static void main(String[] args) {

        // try-catch block to handle exceptions
        try {

            // Create a file object
            File f = new File("C:\\Users\\Sangram\\Documents
                \\JavaDemo");

            // Get all the names of the files
            // present in the given directory

            // Create a FilenameFilter
            FilenameFilter filter = new FilenameFilter() {

                public boolean accept(File f, String name) {
                    return name.startsWith("out");
                }
            };

            // Get all the names of the directory
            // present in the given directory
            // and whose names start with "out"
            File[] DirList = f.listFiles(filter);

            System.out.println("Files/Directories are in the given
                directory:");

            // Display the names of the files/Directories
        }
    }
}
```

```

for (int i = 0; i < DirList.length; i++) {
    System.out.println(DirList[i].getName());
}

catch (Exception e) {

    System.err.println(e.getMessage());
}
}
}

```

Output:

Files/Directories are in the given directory:

output.txt
output1.txt
output2.txt

Example: Java program to demonstrate the use of listFiles(FileFilter f) method to find all the files and directories in a given directory which are text files.

```

import java.io.File;
import java.io.FileFilter;

public class ListingDirectory {

    public static void main(String[] args) {

        // try-catch block to handle exceptions
        try {

            // Create a file object

```

```

File f = new File("C:\\Users\\Sangram\\Documents
                    \\JavaDemo");

// Get all the names of the files
// present in the given directory

// Create a FileFilter
FileFilter filter = new FileFilter() {

    public boolean accept(File f) {
        return f.getName().endsWith("txt");
    }
};

// Get all the names of the files present
// in the given directory
// which are text files
File[] files = f.listFiles(filter);

System.out.println("Files are:");

// Display the names of the files
for (int i = 0; i < files.length; i++) {
    System.out.println(files[i].getName());
}

catch (Exception e) {
    System.err.println(e.getMessage());
}
}
}

```

Output:

Files are:

abc.txt
BinaryInput.txt
BinaryOutput.txt
demofile.txt
MyFile.txt
MyInputFile.txt
MyOutputFile.txt
MyOutputFile1.txt
output.txt
output1.txt
output2.txt

Getting the Directory Roots

Problem

You want to know about the top-level directories, such as C:\ and D:\ on Windows.

Solution

Use the static method `File.listRoots()`.

`listRoots()` method

- The `listRoots()` method is a part of `File` class.
- The `listRoots()` function returns the root directories of all the available file System roots.
- It is guaranteed that the pathname of any file on the system will begin with any one of these roots.
- The function returns **File array**, which contains all the file system roots.

Example: Java program to demonstrate the use of File.listRoots() method.

```
import java.io.File;

public class ListRoot {

    public static void main(String[] args) {

        // Get list of roots names
        File root[] = File.listRoots();

        // check if the root is null or not
        if (root != null) {

            System.out.println("The List of Roots are: ");

            // Print the list
            for (File dr : drives) {
                System.out.println(dr);
            }

        } else {
            System.out.println("There are no roots");
        }
    }
}
```

Output:

The List of Roots are:

C:\

D:\

Using Path instead of File

Problem

- You need more capability than the standard File class. You need to move, copy, delete, and otherwise work on files with a minimum of coding.

Solution

- Consider using the Path class, an intended replacement for File, and the Files class.

java.nio.file.Path

- Path is the particular location of an entity such as file or a directory in a file system so that one can search and access it at that particular location.
- Path is an interface which is introduced in Java NIO file package during Java version 7, and is the representation of location in particular file system.
- As path interface is in Java NIO package so it get its qualified name as `java.nio.file.Path`.
- In general path of an entity could be of two types:
 - absolute path
 - It is the location address from the root to the entity where it locates

- relative path
 - It is the location address which is relative to some other path.

Example:

```
import java.io.File;
import java.nio.file.Path;
import java.nio.file.Paths;

public class PathDemo {

    public static void main(String[] args) {

        //Converts a path string, or a sequence of strings
        //that when joined form a path string, to a Path.
        Path path = Paths.get("C:\\Users\\Sangram\\Documents
                               \\JavaDemo\\abc.txt");
        System.out.println("Relative path: " + path);

        //Returns a Path object representing
        //the absolute path of this path.
        Path absolute = path.toAbsolutePath();
        System.out.println("Absolute path: " + absolute);

        //Returns a name element of this path as a Path object.
        System.out.println("Name: " + path.getFileName());

        //Returns the root component
        System.out.println("Root: " + path.getRoot());

        //Returns the parent path
        System.out.println("Parent: " + path.getParent());
```

```

//Returns the number of name elements in the path.
System.out.println("Name Count: " + path.getNameCount());

//Returns a name element of this path as a Path object.
System.out.println("First Directory: " + path.getName(0));

//Returns a relative Path that is a subsequence
//of the name elements of this path.
System.out.println("Sub Path: " + path.subpath(0, 2));

//Returns the string representation of this path.
System.out.println(path.toString());

// call toFile() to get
// File object from path
File file = path.toFile();

// print file details
System.out.println("File Name:" + file.getName());

if (!file.exists()) {
    System.out.println("This is a path.");
} else {
    System.out.println("This is a file.");
}

//Get the path of file.
System.out.println("Path: " + file.toPath());
}
}

```

Output:

Relative path: C:\Users\Sangram\Documents\JavaDemo\abc.txt

Absolute path: C:\Users\Sangram\Documents\JavaDemo\abc.txt

Name: abc.txt

Root: C:\

Parent: C:\Users\Sangram\Documents\JavaDemo

Name Count: 5

First Directory: Users

Sub Path: Users\Sangram

C:\Users\Sangram\Documents\JavaDemo\abc.txt

File Name:abc.txt

This is a file.

Path: C:\Users\Sangram\Documents\JavaDemo\abc.txt

Chapter 14

Graphical User Interfaces

14.1 Introduction

- ☞ Java has had windowing capabilities since its earliest days. The first version made public was the Abstract Windowing Toolkit, or AWT. Because it used the native toolkit components, AWT was relatively small and simple.
- ☞ The second major implementation was the Swing classes, released in 1998 as part of the Java Foundation Classes. Swing is a full-function, professional-quality GUI toolkit designed to enable almost any kind of client-side GUI-based interaction.
- ☞ AWT lives inside, or rather underneath, Swing, and, for this reason, many programs begin by importing both `java.awt` and `javax.swing`.

***Containers and Components:** There are two types of GUI elements:

- Component: Components are elementary GUI entities, such as Button, Label, and TextField.
- Container: Containers, such as Frame and Panel, are used to hold components in a specific layout (such as FlowLayout or GridLayout). A container can also hold sub-containers.

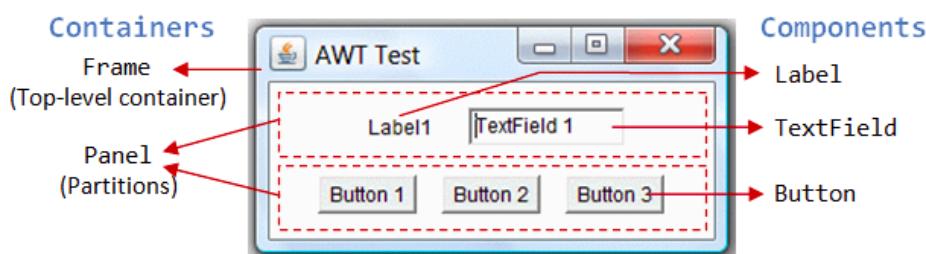


Figure 14.1: Containers and Components

***AWT Container Classes** Each GUI program has a top-level container. The commonly-used top-level containers in AWT are Frame, Dialog and Applet:

- A Frame provides the "main window" for your GUI application. It has a title bar (containing an icon, a title, the minimize, maximize/restore-down and close buttons), an optional menu bar, and the content display area.
- An AWT Dialog is a "pop-up window" used for interacting with the users. A Dialog has a title-bar (containing an icon, a title and a close button) and a content display area, as illustrated.

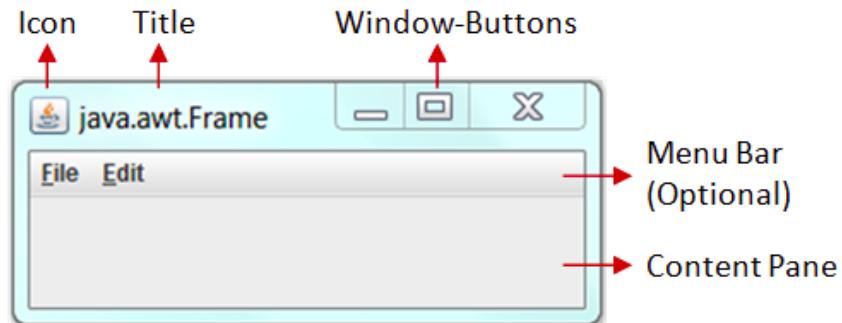


Figure 14.2: AWT Container Classes

14.1.1 Displaying GUI Components

Problem

You want to create some GUI components and have them appear in a window.

Solution

Create a JFrame and add the components to its ContentPane. **Discussion**

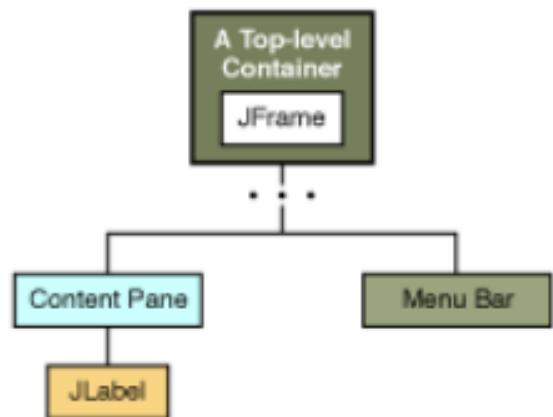


Figure 14.3: GUI

- ☞ The older AWT had a simple Frame component for making main windows; this allowed you to add components directly to it. “Good” programs usually created a panel to fit inside and populate the frame.

- The Swing JFrame is more complex—it comes with not one but two containers already constructed inside it.
- The ContentPane is the main container; you should normally use it as your JFrame's main container.
- The GlassPane has a clear background and sits over the top of the ContentPane ; its primary use is in temporarily painting something over the top of the main ContentPane.
- You can add any number of components (including containers) into this existing container, using the ContentPane add() method.

Example: JFrame Demo:

```
Code:
import java.awt.Container;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class JFrameDemo extends JFrame{
    public JFrameDemo() {
        Container cp = getContentPane();
        // now add Components to Container
        cp.add(new JLabel("A Really Simple Demo",
                JLabel.CENTER));
    }
    public static void main(String[] args) {
        JFrameDemo jfd=new JFrameDemo();
        jfd.setSize(500,100);
        jfd.setTitle("JFrame Demo");
        jfd.setVisible(true);
        jfd.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

Example: JFrame with Quit Button:

```
Code:  
package com.GUI;  
import java.awt.Container;  
import java.awt.FlowLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
public class JFrameDemo2 extends JFrame{  
    JButton quitButton;  
    public JFrameDemo2() {  
        Container cp = getContentPane();  
        // now add Components to Container  
        cp.add(new JLabel("A Really Simple Demo",  
                JLabel.CENTER));  
        cp.setLayout(new FlowLayout());  
        cp.add(quitButton = new JButton("Exit"));  
        quitButton.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                setVisible(false);  
                dispose();  
                System.exit(0);  
            }  
        });  
    }  
    public static void main(String[] args) {  
        JFrameDemo2 jfd=new JFrameDemo2();  
        jfd.setSize(500,100);  
        jfd.setTitle("JFrame Demo");  
        jfd.setVisible(true);  
        jfd.setDefaultCloseOperation(EXIT_ON_CLOSE);  
    }  
}
```

14.1.2 Run Your GUI on the Event Dispatching Thread

Problem

Your application fails to start, with a message like Running on UI thread when not expected . Or, your application crashes very sporadically.

Solution

Run your UI on the UI thread, which Java names the “Event Dispatching Thread” or EDT.

Example: Dialog box message demo

```
package com.GUI;
import javax.swing.JOptionPane;
import javax.swing.SwingUtilities;
public class MessageDisplay {
    public static void main(String[] args) throws Exception {
        System.out.println("Program quite during the Run the
                           main()");
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    JOptionPane.showMessageDialog(null, "Do You Want To
                           Exit");
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

Assignment:

1. Demonstrate the use of GUI Components

Solution:

```
Panel pnl = new Panel();           // Panel is a container
Button btn = new Button("Press");   // Button is a component
pnl.add(btn);                     // The Panel container adds a Button component
```

14.1.3 Designing a Window Layout

Problem

The default layout isn't good enough.

Solution

Learn to deal with a layout manager.

Discussion

- The container classes such as Panel have the capability to contain a series of components, but you can arrange components in a window in many ways. Rather than clutter up each container with a variety of different layout computations, the designers of the Java API used a sensible design pattern to divide the labor.
- A layout manager is an object that performs the layout computations for a container. The AWT package has five common layout manager classes, and Swing has a few more.
- If your JFrame is full of well-behaved components, you can set its size to be “just the size of all included components, plus a bit for padding,” just by calling the pack() method, which takes no arguments.
- The pack() method goes around and asks each embedded component for its preferred size.

- The JFrame is then set to the best size to give the components their preferred sizes as much as is possible. If not using pack() , you need to call the setSize() method, which requires either a width and a height, or a Dimension object containing this information.
- JFrameDemo in one of two ways: either we can use a FlowLayout or specify BorderLayout regions for the label and the button.

Example: Layout Manager Demo

```

package com.GUI;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import java.awt.Container;
import java.awt.FlowLayout;
public class JFrameLayout extends JFrame{
    public JFrameLayout() {
        Container cp = getContentPane();
        // Make sure it has a FlowLayout layout manager.
        cp.setLayout(new FlowLayout());
        // now add Components to container
        cp.add(new JLabel("FlowLayout manager"));
        cp.add(new JButton("Yes!"));
        pack(); //frame size or preference size of frame
    }
    public static void main(String[] args) {
        JFrameLayout jfd=new JFrameLayout();
        jfd.setVisible(true);
        jfd.setSize(300,100);
        jfd.setTitle("JFrame Layout");
        jfd.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}

```

14.1.4 A Tabbed View of Life

Problem

These layouts don't include a tab layout, and you need one.

Solution

Use a JTabbedPane .

Discussion

The JTabbedPane class acts as a combined container and layout manager. It implements a conventional tab layout.

Example: Tab Demo

```
Code:  
package com.GUI;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JTabbedPane;  
public class TabPaneDemo {  
    protected JTabbedPane tabPane;  
    public TabPaneDemo() {  
        tabPane = new JTabbedPane();  
        tabPane.add(new JLabel("One", JLabel.CENTER), "First");  
        tabPane.add(new JLabel("Two", JLabel.CENTER), "Second");  
        tabPane.add(new JLabel("Three", JLabel.CENTER),  
                   "third");  
    }  
    public static void main(String[] a) {  
        JFrame f = new JFrame("Tab Demo");  
        f.getContentPane().add(new TabPaneDemo().tabPane);  
        f.setSize(300, 200);  
        f.setVisible(true);  
    }  
}
```

14.1.5 Action Handling: Making Buttons Work

Problem

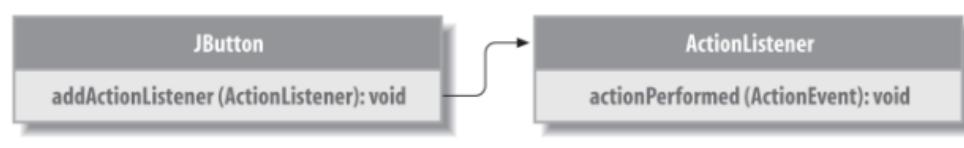
Your button doesn't do anything when the user presses it.

Solution

Add an ActionListener to do the work.

Discussion

AWT listener relationships



Example: pushing a button causes the program to print a friendly message

```
package com.GUI;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
/** Demonstrate simple use of Button */
public class ButtonDemo extends JFrame implements
    ActionListener {
    public ButtonDemo() {
        setLayout(new FlowLayout());
        JButton b1=new JButton("A button");
        add(b1);
        b1.addActionListener(this);
        setSize(300, 200);
    }
    public void actionPerformed(ActionEvent event) {
        System.out.println("Thanks for pushing my button!");
    }
    public static void main(String[] unuxed) {
        new ButtonDemo().setVisible(true);
    }
}
```

Assignment

1. Demonstrate the use of JTabbedPane.
2. Demonstrate the BorderLayout to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

Hints

```
public static final int NORTH
public static final int SOUTH
public static final int EAST
public static final int WEST
public static final int CENTER
```

14.1.6 Action Handling Using Lambdas

Problem

You want to use Java 8's lambda expressions to simplify GUI programming.

Solution

Write the lambda expression as the argument to, for example,
JComponent.addActionListener()

Syntax: (lambda operator) -> body (p) -> System.out.println("One parameter:" + p);

Example: Demonstrate a JButton with Lambda Action Listeners

```
Code:  
package com.GUI;  
import java.awt.FlowLayout;  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JOptionPane;  
public class ButtonDemo3L extends JFrame{  
    public ButtonDemo3L() { //constructor  
        super("ButtonDemo Lambda");  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new FlowLayout());  
        JButton b;  
        add(b = new JButton("A button"));  
        // Minimalist style  
        b.addActionListener(e ->  
            JOptionPane.showMessageDialog(this,  
                "Thanks for pushing my first button!"));  
        add(b = new JButton("Another button"));  
        // Longer style, with {} around body.  
        b.addActionListener(e ->  
            {JOptionPane.showMessageDialog(this,  
                "Thanks for pushing my second button!");}  
        );  
        pack();  
    }  
    public static void main(String[] args) {  
        new ButtonDemo3L().setVisible(true);  
    }  
}
```

14.1.7 Terminating a Program with “Window Close”

Problem

Nothing happens when you click the close button on the title bar of an AWT Frame . When you do this on a Swing JFrame , the window disappears but the application does not exit.

Solution

Use JFrame’s **setDefaultCloseOperation()** method or **add a WindowListener** and have it exit the application.

Discussion

The Swing JFrame has a **setDefaultCloseOperation()** method, which controls the default behavior. You can pass it one of the values defined in the Swing WindowConstants class:

WindowConstants.DO NOTHING ON CLOSE

Ignore the request. The window stays open. Useful for critical dialogs; probably antisocial for most “main application” type windows.

WindowConstants.HIDE ON CLOSE

Hide the window (default).

WindowConstants.DISPOSE ON CLOSE

Hide and dispose the window.

WindowConstants.EXIT_ON_CLOSE

Exit the application on close, obviating the need for a WindowListener ! Does not give you a chance to save data; for that, you need a **WindowListener**.

Assignment

1. Demonstrate JTextArea class that display text.

Hints

```
public class JTextArea extends JTextComponent
//main code for JTextArea
JFrame f= new JFrame();
JTextArea area=new JTextArea("Welcome to
javatpoint");
area.setBounds(10,30, 200,200);
f.add(area);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
```

2. Program to demonstrate JRadioButton.

```
public class JRadioButton extends JToggleButton implements Accessible
```

3. Set frame.dispose() on the click of a button to close JFrame.

```
JFrame frame = new JFrame();
JButton button = new JButton("Click to Close!");
```

Close the JFrame on the click of the above button with Action Listener:

```
button.addActionListener(e ->{ frame.dispose(); });
```

Example: Class WindowDemo puts up a JFrame and closes when you ask it to.

```
Code:  
package com.GUI;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import javax.swing.JFrame;  
import javax.swing.WindowConstants;  
public class WindowDemo extends JFrame{  
    public static void main(String[] argv) {  
        JFrame f = new WindowDemo();  
        f.setVisible(true);  
    }  
    public WindowDemo() {  
        setSize(200, 100);  
        setDefaultCloseOperation(WindowConstants.DO_NOTHING_ON_CLOSE);  
        addWindowListener(new WindowDemoAdapter());  
    }  
    /** Named Inner class that closes a Window. */  
    class WindowDemoAdapter extends WindowAdapter {  
  
        public void windowClosing(WindowEvent e) {  
            // whimsy - close randomly, ~ 1 times in 3  
            if (Math.random() > 0.666) {  
                System.out.println("Goodbye!");  
                WindowDemo.this.setVisible(false); //  
                window will close  
                WindowDemo.this.dispose(); // and be freed  
                up.  
                System.exit(0);  
            }  
            System.out.println("You asked me to close, but not to I  
            chose.");  
        }  
    }  
}  
  
Output:  
You asked me to close, but not to I chose.  
Goodbye!
```

14.1.8 Dialogs: When Later Just Won't Do

Problem

You need a bit of feedback from the user right now.

Solution: Use a **JOptionPane** method to show a prebuilt dialog. Or subclass **JDialog**.

Example

```
Code:  
package com.GUI;  
import java.awt.Container;  
import java.awt.FlowLayout;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JOptionPane;  
public class JOptionPaneDemo extends JFrame{  
    // Constructor  
    JOptionPaneDemo(String s) {  
        super(s);  
        Container cp = getContentPane();  
        cp.setLayout(new FlowLayout());  
        JButton b = new JButton("A button"); //one button  
        b.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                JOptionPane.showMessageDialog(  
                    JOptionPaneDemo.this,  
                    "dialog1.text",  
                    "dialog1.title",  
                    JOptionPane.INFORMATION_MESSAGE);  
            }});  
        cp.add(b); //other button  
        b = new JButton("GoodBye");  
        b.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.exit(0);  
            }});  
        cp.add(b);  
        setSize(200, 150); // the main window  
        pack();  
    }  
    public static void main(String[] arg) {  
        JOptionPaneDemo x = new JOptionPaneDemo("Testing 1 2 3...");  
        x.setVisible(true);  
    }  
}
```

14.1.9 Choosing a Value with JSpinner

Problem

You want to let the user choose from a fixed set of values, but do not want to use a Jlist or JComboBox because they take up too much “screen real estate.”

Solution

Use a JSpinner .

Discussion

The JSpinner class lets the user click up or down to cycle through a set of values. The values can be of any type because they are managed by a helper of type SpinnerModel and displayed by another helper of type SpinnerEditor . A series of predefined SpinnerModels handle Number's, Date's, and List's (which can be arrays or Collections).

Example: Jspinner Demo

```
Code:  
package com.GUI;  
import java.awt.Container;  
import java.awt.GridLayout;  
import javax.swing.JFrame;  
import javax.swing.JSpinner;  
import javax.swing.SpinnerDateModel;  
import javax.swing.SpinnerListModel;  
  
public class SpinnerDemo {  
    public static void main(String[] args) {  
        JFrame jf = new JFrame("It Spins");  
        Container cp = jf.getContentPane();  
        cp.setLayout(new GridLayout(0,1));  
  
        // Create a JSpinner using one of the pre-defined  
        // SpinnerModels  
        JSpinner dates = new JSpinner(new SpinnerDateModel());  
        cp.add(dates);  
  
        // Create a JSpinner using a SpinnerListModel.  
        String[] data = { "One", "Two", "Three" };  
        JSpinner js = new JSpinner(new SpinnerListModel(data));  
        cp.add(js);  
        jf.setSize(200, 80);  
        jf.setVisible(true);  
    }  
}
```

14.1.10 Choosing a File with JFileChooser

Problem

You want to allow the user to select a file by name using a traditional windowed file dialog.

Solution

Use a JFileChooser .

Discussion

- The JFileChooser dialog provides a fairly standard file chooser. It has elements of both a Windows chooser and a Mac chooser, with more resemblance to the former than the latter.
- If you want to have control over which files appear, you need to provide one or more FileFilter subclasses.
- Each FileFilter subclass instance passed into the JFileChooser's addChoosableFileFilter() method becomes a selection in the chooser's Files of Type: choice. The default is All Files (.).

Example: Choosing a File

Code :

```
package com.GUI;
import java.io.File;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class FileChooseDemo extends JPanel{
    public FileChooseDemo(JFrame f) {
        final JFrame frame = f;
        final JFileChooser chooser = new JFileChooser();
        JButton b = new JButton("Choose file...");
        add(b);
        b.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                int returnVal = chooser.showOpenDialog(frame);
                if (returnVal == JFileChooser.APPROVE_OPTION) {
                    File file = chooser.getSelectedFile();
                    System.out.println("You chose a " +
                        (file.isFile() ? "file" : "directory") +
                        " named: " + file.getPath());
                } else {
                    System.out.println("You did not choose a
                        filesystem object.");
                }
            }
        });
    }
    public static void main(String[] args) {
        JFrame f = new JFrame("JFileChooser Demo");
        f.getContentPane().add(new FileChooseDemo(f));
        f.pack();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Output:

```
You chose a file named: /home/chukhu/Desktop/JavadocDemo.java
You did not choose a filesystem object.
```

Assignment

1. Demonstrate use of JOptionPane.

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. **public class JOptionPane extends JComponent implements Accessible**

Solution

```
import javax.swing.*;
public class OptionPaneExample {
    JFrame f;
    OptionPaneExample(){f=new JFrame();
    String name= JOptionPane.showInputDialog(f, "Enter Name");}
    public static void main(String[] args) {new
        OptionPaneExample();}}
```

2. Demonstrate JScrollPane.

The object of JScrollPane class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

public class JScrollPane extends JComponent implements Adjustable, Accessible

14.1.11 Choosing a Color

Problem

You want to allow the user to select a color from all the colors available on your computer.

Solution: Use Swing's JColorChooser .

Discussion:

- Construct it and place it in a panel.
- Call its createDialog() and get a JDialog back.
- Call its showDialog() and get back the chosen color.

Methods of operating the chooser:

- Swatches mode: The user can pick from one of a few hundred color variants.
- HSB mode: The user picks one of Hue, Saturation, or Brightness, a standard way of representing color value. The user can adjust each value by slider.
- RGB mode: The user picks Red, Green, and Blue components by sliders.

Example

```
Code:
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JColorChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
public class JColorChooserDemo extends JFrame {
/** A canvas to display the color in. */
protected JLabel demo;
/** Constructor - set up the entire GUI for this program */
public JColorChooserDemo() {
super("Swing Color Demo illustrated");
Container cp = getContentPane();
JButton jButton;
cp.add(jButton = new JButton("Change Color..."),
BorderLayout.NORTH); //button
jButton.setToolTipText(""); //text
jButton.addActionListener(new ActionListener() { //action
on button
public void actionPerformed(ActionEvent actionEvent)
{
    Color ch = JColorChooser.showDialog( //show the
        dialog of color
        JColorChooserDemo.this,
// parent
        "Swing Demo Color Popup",
//title
        demo.getForeground()); //color in which text is shown
    System.out.println("Your selected color is " + ch);
    //default
if (ch != null) {
    demo.setForeground(ch);
    demo.repaint(); //repaint the color of text
}}});
cp.add(BorderLayout.CENTER, demo =
    new JLabel("Your One True Color", JLabel.LEFT));
demo.setToolTipText("This is the last color you
choose");
pack();
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
public static void main(String[] argv) {
new JColorChooserDemo().setVisible(true);
}}
Output:
Your selected color is java.awt.Color[r=255,g=51,b=51]
Your selected color is java.awt.Color[r=153,g=204,b=0]
Your selected color is java.awt.Color[r=255,g=0,b=51]
```

14.1.12 Formatting Jcomponents with HTML

Problem

You want more control over the formatting of text in JLabel and friends. **Solution**
Use HTML in the text of the component.

Discussion

The Swing components that display text, such as `JLabel`, format the text as HTML—instead of as plain text—if the first six characters are the obvious tag `<html>`. The program `JLabelHTMDemo` just puts up a `JLabel` formatted using this Java code:

Formatting Jcomponents with HTML

```
Code:  
package Graphics;  
import java.awt.BorderLayout;  
import java.awt.Container;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import javax.swing.JButton;  
import javax.swing.JFrame;  
  
public class JLabelHTMDemo extends JFrame{  
    /** Construct the object including its GUI */  
    public JLabelHTMDemo() {  
        super("JLabelHTMDemo");  
        Container cp = getContentPane();  
        JButton component = new JButton(  
            "<html>" +  
            "<body bgcolor='white'>" +  
            "<h1><font color='red'>Welcome</font></h1>" +  
            "<p>This button will be formatted according to the usual " +  
            "HTML rules for formatting of paragraphs.</p>" +  
            "</body></html>");  
        component.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent evt) {  
                System.out.println("Thank you!");  
            }  
        });  
        cp.add(BorderLayout.CENTER, component);  
        setSize(200, 400);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    }  
    public static void main(String[] args) {  
        new JLabelHTMDemo().setVisible(true);  
    }  
}
```

Assignment

1. Using a `JLabel` and pass in HTML for the text.

Centering a Main Window:

```
JFrame frame = new JFrame();  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
JLabel label = new JLabel("<html>bold <br>  
    plain</html>");  
frame.add(label);  
frame.setSize(300, 200);  
frame.setVisible(true);
```

14.1.13 Centering a Main Window:

Problem

You want to change the look and feel of an application.

Solution

Use the static UIManager.setLookAndFeel() method. Maybe.

Centering a Main Window:

```
Code:  
package Graphics;  
import javax.swing.SwingUtilities;  
import javax.swing.UIManager;  
public class LNFSwitcher {  
    protected JFrame theFrame; /* The frame. */  
    protected Container cp; /* Its content pane */  
    /* Start with the Java look-and-feel, if possible */  
    final static String PREFERREDLOOKANDFEELNAME =  
        "javax.swing.plaf.metal.MetalLookAndFeel";  
    protected String curLF = PREFERREDLOOKANDFEELNAME;  
    protected JRadioButton previousButton;  
    /* Construct a program... */  
    public LNFSwitcher() {  
        super();  
        theFrame = new JFrame("LNF Switcher");  
        theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        cp = theFrame.getContentPane();  
        cp.setLayout(new FlowLayout());  
        ButtonGroup bg = new ButtonGroup();  
        JRadioButton bJava = new JRadioButton("Java");  
        bJava.addActionListener(new LNFSetter(  
            "javax.swing.plaf.metal.MetalLookAndFeel", bJava));  
        bg.add(bJava);  
        cp.add(bJava);  
        JRadioButton bMSW = new JRadioButton("MS-Windows");  
        bMSW.addActionListener(new LNFSetter(  
            "com.sun.java.swing.plaf.windows.WindowsLookAndFeel",  
            bMSW));  
        bg.add(bMSW);  
        cp.add(bMSW);  
        JRadioButton bMotif = new JRadioButton("Motif");  
        bMotif.addActionListener(new LNFSetter(  
            "com.sun.java.swing.plaf.motif.MotifLookAndFeel",  
            bMotif));  
        bg.add(bMotif);  
        cp.add(bMotif);
```

continue...

```
JRadioButton bMac = new JRadioButton("Sun-MacOS");
bMac.addActionListener(new LNFSetter(
    "com.sun.java.swing.plaf.mac.MacLookAndFeel", bMac));
bg.add(bMac);
cp.add(bMac);
String defaultLookAndFeel =
    UIManager.getSystemLookAndFeelClassName();
// System.out.println(defaultLookAndFeel);
JRadioButton bDefault = new JRadioButton("Default");
bDefault.addActionListener(new LNFSetter(
    defaultLookAndFeel, bDefault));
bg.add(bDefault);
cp.add(bDefault);
(previousButton = bDefault).setSelected(true);
theFrame.pack();
/* Class to set the Look and Feel on a frame */
class LNFSetter implements ActionListener {
String theLNFName;
JRadioButton thisButton;
/** Called to setup for button handling */
LNFSetter(String lnfName, JRadioButton me) {
    theLNFName = lnfName;
    thisButton = me;
}
/** Called when the button actually gets pressed. */
public void actionPerformed(ActionEvent e) {
    try {
        UIManager.setLookAndFeel(theLNFName);
        SwingUtilities.updateComponentTreeUI(theFrame);
        theFrame.pack();
    } catch (Exception evt) {
        JOptionPane.showMessageDialog(null,
            "setLookAndFeel didn't work: " + evt,
            "UI Failure", JOptionPane.INFORMATION_MESSAGE);
        previousButton.setSelected(true); // reset the GUI
        to agree
    }
    previousButton = thisButton;
}}
public static void main(String[] argv) {
LNFSwitcher o = new LNFSwitcher();
o.theFrame.setVisible(true);
}}
```

14.1.14 Program : Custom Front Chooser

Problem

You want to allow the user to select a font, but standard Java doesn't yet include a Font Chooser dialog. **Solution**

Use my FontChooser dialog class.

Discussion

As we saw in Recipe 12.3, you can manually select a font by calling the java.awt.Font class constructor, passing in the name of the font, the type you want (plain, bold, italic, or bold+italic), and the point size: `Font f = new Font("Helvetica", Font.BOLD, 14); setfont(f);`

Custom Front Chooser

```
Code:  
public class FontChooser extends JDialog {  
    private static final long serialVersionUID =  
        5363471384675038069L;  
    public static final String DEFAULT_TEXT = "Lorem ipsum dolor";  
    /** The font the user has chosen */  
    protected Font resultFont = new Font("Serif", Font.PLAIN, 12);  
    /** The resulting font name */  
    protected String resultName;  
    /** The resulting font size */  
    protected int resultSize;  
    /** The resulting boldness */  
    protected boolean isBold;  
    /** The resulting italicness */  
    protected boolean isItalic;  
    /** Display text */  
    protected String displayText = DEFAULT_TEXT;  
    /** The font name chooser */  
    protected JList fontNameChoice;  
    /** The font size chooser */  
    protected JList fontSizeChoice;  
    /** The bold and italic choosers */  
    JCheckBox bold, italic;  
    /** The list of font sizes */  
    protected Integer fontSizes[] = {  
        8, 9, 10, 11, 12, 14, 16, 18, 20, 24, 30, 36, 40, 48, 60, 72  
    };  
    /** The index of the default size (e.g., 14 point == 4) */
```

continue...

```
protected static final int DEFAULT_SIZE = 4;
/** The font display area.*/
protected JLabel previewArea;
/** Construct a FontChooser -- Sets title and gets
 * array of fonts on the system. Builds a GUI to let
 * the user choose one font at one size.
*/
public FontChooser(JFrame f) {
super(f, "Font Chooser", true);
Container cp = getContentPane();
JPanel top = new JPanel();
top.setBorder(new TitledBorder(new EtchedBorder(), "Font"));
top.setLayout(new FlowLayout());
String[] fontList =
    GraphicsEnvironment.getLocalGraphicsEnvironment().
getAvailableFontFamilyNames();
fontNameChoice = new JList(fontList);
top.add(new JScrollPane(fontNameChoice));
fontNameChoice.setVisibleRowCount(fontSizes.length);
fontNameChoice.setSelectedValue("Serif", true);
fontSizeChoice = new JList(fontSizes);
top.add(fontSizeChoice);
fontSizeChoice.setSelectedIndex(fontSizes.length * 3 / 4);
cp.add(top, BorderLayout.NORTH);
JPanel attrs = new JPanel();
top.add(attrs);
attrs.setLayout(new GridLayout(0,1));
attrs.add(bold =new JCheckBox("Bold", false));
attrs.add(italic=new JCheckBox("Italic", false));
// Make sure that any change to the GUI will trigger a font
// preview.
ListSelectionListener waker = new ListSelectionListener() {
public void valueChanged(ListSelectionEvent e) {
previewFont();
}};
bold.addItemListener(waker2);
italic.addItemListener(waker2);
previewArea = new JLabel(displayText, JLabel.CENTER);
previewArea.setSize(200, 50);
```

continue...

```
cp.add(previewArea, BorderLayout.CENTER);
JPanel bot = new JPanel();
JButton okButton = new JButton("Apply");
bot.add(okButton);
okButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
previewFont();
dispose();
setVisible(false);
}});
JButton canButton = new JButton("Cancel");bot.add(canButton);
canButton.addActionListener(new ActionListener() {
public void actionPerformed(ActionEvent e) {
// Set all values to null. Better: restore previous.
resultFont = null;
resultName = null;
resultSize = 0;
isBold = false;
isItalic = false;
dispose();
setVisible(false);
}});
cp.add(bot, BorderLayout.SOUTH);
previewFont(); // ensure view is up to date!
pack();
setLocation(100, 100);}
/** Called from the action handlers to get the font info,
 * build a font, and set it.*/
protected void previewFont() {
resultName = (String)fontNameChoice.getSelectedItem();
String resultSizeName =
    fontSizeChoice.getSelectedItem().toString();
int resultSize = Integer.parseInt(resultSizeName);
isBold = bold.isSelected();
isItalic = italic.isSelected();
int attrs = Font.PLAIN;
if (isBold) attrs = Font.BOLD;
if (isItalic) attrs |= Font.ITALIC;
resultFont = new Font(resultName, attrs, resultSize);
// System.out.println("resultName = " + resultName + "; " +
// "resultFont = " + resultFont);
previewArea.setFont(resultFont);
pack();}
/** Retrieve the selected font name. */
public String getSelectedName() {
return resultName; }
```

continue...

```
/** Retrieve the selected size */
public int getSelectedSize() {
    return resultSize;
}
/** Retrieve the selected font, or null */
public Font getSelectedFont() {
    return resultFont;
}
public String getDisplayText() {
    return displayText;
}
public void setDisplayText(String displayText) {
    this.displayText = displayText;
    previewArea.setText(displayText);
    previewFont();
}
public JList getFontNameChoice() {
    return fontNameChoice;
}
public JList getSizeChoice() {
    return fontSizeChoice;
}
public boolean isBold() {
    return isBold;
}
public boolean isItalic() {
    return isItalic;
}
}
```

Assignment

1. Create Edit menu for Notepad: Using JMenuBar, JMenu and JMenuItem.
2. Program to select a color from all colors available in computer using Swing's JColorChooser.
3. Demonstrate the slider by using JSlider, a user can select a value from a specific range.

Solution

```
import javax.swing.*;
public class SliderExample extends JFrame{
    public SliderExample() {
        JSlider slider = new JSlider(JSlider.HORIZONTAL , 0,
            50, 25);
        slider.setMinorTickSpacing(2);
        slider.setMajorTickSpacing(10);
        slider.setPaintTicks(true);
        slider.setPaintLabels(true);
        JPanel panel=new JPanel();
        panel.add(slider);
        add(panel);
    }
    public static void main(String s[]) {
        SliderExample frame=new SliderExample();
        frame.pack();
        frame.setVisible(true);
    }
}
```

4. Program to use JFileChooser to select directory only.

Hints

```
chooser = new JFileChooser();
chooser.setCurrentDirectory(new java.io.File("."));
chooser.setDialogTitle(choosertitle);
chooser.setFileSelectionMode(JFileChooser.DIRECTORIES_ONLY);
// disable the "All files" option.
chooser.setAcceptAllFileFilterUsed(false);
if (chooser.showOpenDialog(this) ==
    JFileChooser.APPROVE_OPTION){
    System.out.println("getCurrentDirectory(): "
        + chooser.getCurrentDirectory());
    System.out.println("getSelectedFile() : "
        + chooser.getSelectedFile());
}
else {
    System.out.println("No Selection ");
}}
```

CHAPTER 13

Network Clients

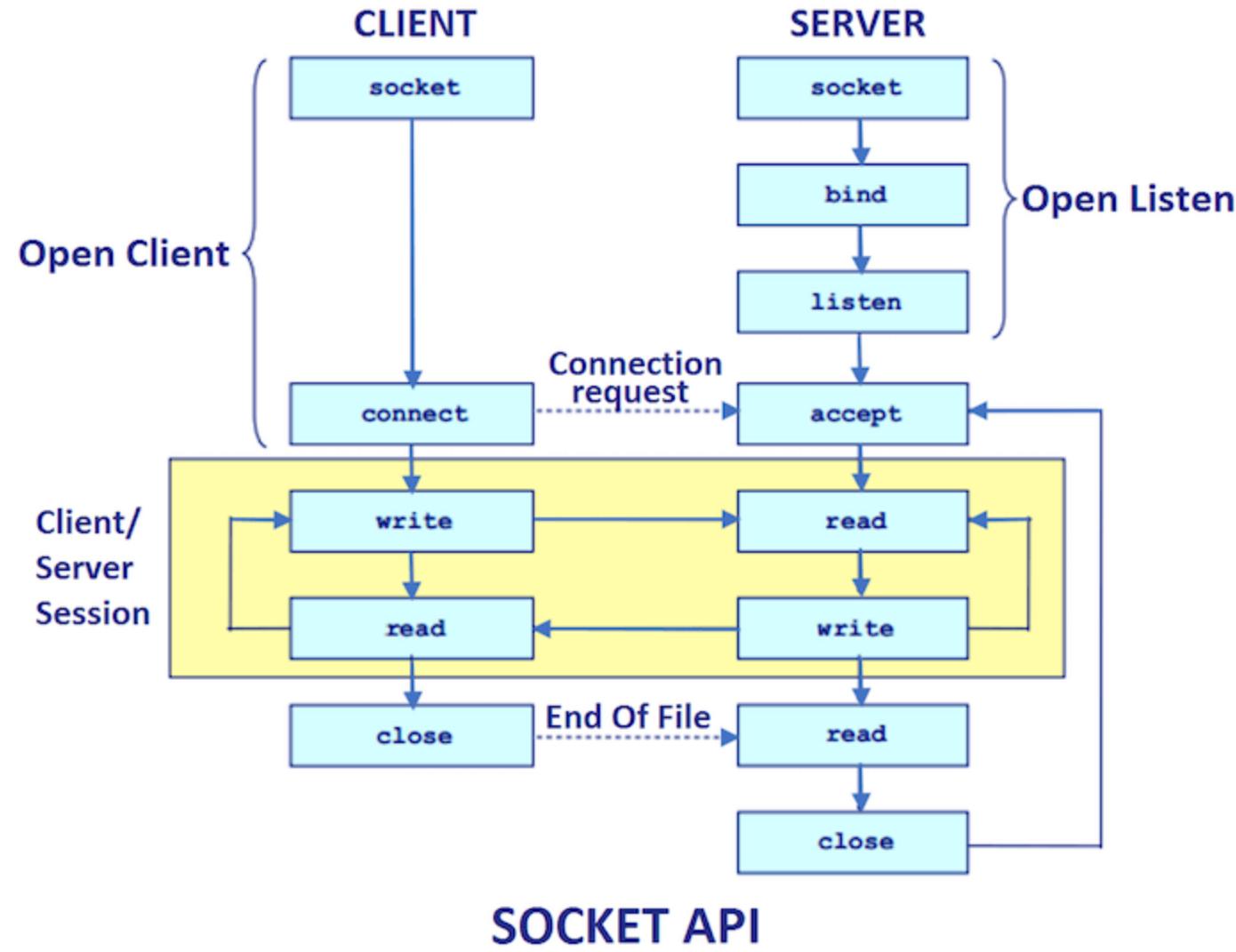
Introduction

- Sockets allow communication between two different processes on the same or different machines.
- Java Socket programming is used for communication between the applications running on different JRE.
- There are two communication protocols that one can use for socket programming:
 - Transfer Control Protocol (TCP)
 - User Datagram Protocol (UDP)
- TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.
- TCP is reliable as it guarantees delivery of data to the destination router.
- UDP is the connection-less protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection.
- The delivery of data to the destination cannot be guaranteed in UDP.
- This lecture presents connection oriented socket programming.
- The two key classes from the `java.net` package used for connection-oriented socket programming
 - `Socket`
 - `ServerSocket`

Introduction(Continue..)

The following steps occur when establishing a TCP connection between two computers using sockets

1. The server instantiates a ServerSocket object, denoting which port number communication is to occur on.
2. The server invokes the accept() method of the ServerSocket class. This method waits until a client connects to the server on the given port.
3. After the server is waiting, a client instantiates a Socket object, specifying the server name and the port number to connect to.
4. The constructor of the Socket class attempts to connect the client to the specified server and the port number. If communication is established, the client now has a Socket object capable of communicating with the server.
5. On the server side, the accept() method returns a reference to a new socket on the server that is connected to the client's socket.



Contacting a Server

Problem

You need to contact a server using TCP/IP.

Solution

Just create a `java.net.Socket`, passing the hostname and port number into the Constructor.

Discussion

When creating a socket, you pass in the hostname and the port number. The `java.net.Socket` constructor does the `gethostbyname()` and the `socket()` system call, sets up the server's `sockaddr_in` structure, and executes the `connect()` call. All you have to do is catch the errors, which are subclassed from the familiar `IOException`

```
import java.net.Socket;

/* Client with NO error handling */
public class ConnectSimple {

    public static void main(String[] argv) throws Exception {

        try (Socket sock = new Socket("localhost", 8080)) {

            /* If we get here, we can read and write on the socket "sock" */
            System.out.println(" *** Connected OK ***");

            /* Do some I/O here... */
        }
    }
}
```

Finding and Reporting Network Addresses

Problem

You want to look up a host's address name or number or get the address at the other end of a network connection.

Solution

Get an InetAddress object.

Discussion

- The InetAddress object represents the Internet address of a given computer or host.
- It has no public constructors; you obtain an InetAddress by calling the static `getByName()` method, passing in either a hostname like `www.darwinsys.com` or a network address as a string, like `1.23.45.67`.
- All the “lookup” methods in this class can throw the checked `UnknownHostException` (a subclass of `java.io.IOException`), which must be caught or declared on the calling method’s header.
- The method `getHostAddress()` gives you the numeric IP address (as a string) corresponding to the InetAddress.
- The `getHostName()`, reports the name of the InetAddress. This can be used to print the address of a host given its name, or vice versa.

```
//import java.net.Socket;
import java.io.*;
import java.net.*;

public class InetAddrDemo {
    public static void main(String[] args) throws IOException {
        String hostName = "www.darwinsky.com";
        String ipNumber = "8.8.8.8"; // currently a well-known Google DNS server

        // Show getting the InetAddress (looking up a host) by host name
        System.out.println(hostName + "s address is " + InetAddress.getByName(hostName).getHostAddress());
        // Look up a host by address
        System.out.println(ipNumber + "s name is " + InetAddress.getByName(ipNumber).getHostName());
        // Look up my localhost addresss
        final InetAddress localHost = InetAddress.getLocalHost();
        System.out.println("My localhost address is " + localHost);

        // Show getting the InetAddress from an open Socket
        String someServerName = "www.google.com";
        // assuming there's a web server on the named server:
        Socket theSocket = new Socket(someServerName, 80);
        InetAddress remote = theSocket.getInetAddress();
        System.out.printf("The InetAddress for %s is %s%n", someServerName, remote);
    }
}
```

Handling Network Errors

Problem

You want more detailed reporting than just IOException if something goes wrong.

Solution

Catch a greater variety of exception classes. SocketException has several subclasses; the most notable are:

ConnectException : the connection was refused by the machine at the other end (the server machine)

NoRouteToHostException: completely explains the failure.

```
import java.io.*;
import java.net.*;
public class ConnectFriendly {
    public static void main(String[] argv) {
        String server_name = argv.length == 1 ? argv[0] : "localhost";
        int tcp_port = 80;

        try (Socket sock = new Socket(server_name, tcp_port)) {
            /* If we get here, we can read and write on the socket.*/
            System.out.println(" *** Connected to " + server_name + " ***");
            /* Do some I/O here... */
        } catch (UnknownHostException e) {
            System.err.println(server_name + " Unknown host");
            return;
        } catch (NoRouteToHostException e) {
            System.err.println(server_name + " Unreachable" );
            return;
        } catch (ConnectException e) {
            System.err.println(server_name + " connect refused");
            return;
        } catch (java.io.IOException e) {
            System.err.println(server_name + ' ' + e.getMessage());//return detailed message of this throwable instance
            return;
        }
    }
}
```

Reading and Writing Textual Data

Problem

Having connected, you wish to transfer textual data.

Solution

Construct a BufferedReader or PrintWriter from the socket's getInputStream() or GetOutputStream().

Discussion

The Socket class has methods that allow you to get an InputStream or OutputStream to read from or write to the socket. It has no method to fetch a Reader or Writer.

You can always create a Reader from an InputStream or a Writer from an OutputStream using the conversion classes.

The paradigm for the two most common forms is:

```
BufferedReader is = new BufferedReader( new InputStreamReader(sock.getInputStream( )));
PrintWriter os = new PrintWriter(sock.getOutputStream( ), true);
```

The **getInputStream()** method of Java Socket class returns an input stream for the given socket.

The **getOutputStream()** method of Java Socket class returns an output stream for the given socket.

PrintWriter class Prints formatted representations of objects to a text-output stream. It implements all of the print methods found in PrintStream.

Following example reads a line of text from the “daytime” service, which is offered by fullfledged TCP/IP suites (such as those included with most Unixes). You don’t have to send anything to the Daytime server; you simply connect and read one line. The server writes one line containing the date and time and then closes the connection.

DaytimeText.java

```
public class DaytimeText {  
    public static final short TIME_PORT = 13; // port 13 is for Daytime protocol  
    public static void main(String[] argv) {  
        String hostName;  
        if (argv.length == 0)  
            hostName = "localhost";  
        else  
            hostName = argv[0];  
        try {  
            Socket sock = new Socket(hostName, TIME_PORT);  
            BufferedReader is = new BufferedReader(new InputStreamReader(sock.getInputStream()));  
            String remoteTime = is.readLine();  
            System.out.println("Time on " + hostName + " is " + remoteTime);  
        } catch (IOException e) {  
            System.err.println(e);  
        }  
    }  
}
```

The second example, shows both reading and writing on the same socket. The Echo server simply echoes back whatever lines of text you send it. It helps in network testing and also in testing clients of this type. The converse() method holds a short conversation with the Echo server on the named host; if no host is named, it tries to contact localhost.

EchoClientOneLine.java

```
public class EchoClientOneLine {
    /** What we send across the net */
    String mesg = "Hello across the net";
    public static void main(String[] argv) {
        if (argv.length == 0)
            new EchoClientOneLine().converse("localhost");
        else
            new EchoClientOneLine().converse(argv[0]);
    }
    /** Hold one conversation across the net */
    protected void converse(String hostName) {
        try {
            Socket sock = new Socket(hostName, 7); // echo server.
            BufferedReader is = new BufferedReader(new InputStreamReader(sock.getInputStream()));
            PrintWriter os = new PrintWriter(sock.getOutputStream(), true);
            // Do the CRLF ourself since println appends only a \r on platforms where that is the native line
            ending.
            os.print(mesg + "\r\n"); // Carriage Return and Line Feed, or CRLF.
            os.flush();
            String reply = is.readLine();
            System.out.println("Sent \" + mesg + "\"");
            System.out.println("Got \" + reply + "\"");
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Reading and Writing Serialized Data

Problem

Having connected, you wish to transfer serialized object data.

Solution

Construct an `ObjectInputStream` or `ObjectOutputStream` from the socket's `getInputStream()` or `getOutputStream()`.

Object serialization is the ability to convert in memory objects to an external form that can be sent serially (a byte at a time). This program (and its server) operate one service that isn't normally provided by TCP/ IP, because it is Java-specific.

You can find the server for this program in Serside Side chapter. This shows the client code.

DaytimeObject.java

```
public class DaytimeObject {  
    /** The TCP port for the object time service. */  
    public static final short TIME_PORT = 1951;  
    public static void main(String[] argv) {  
        String hostName;  
        if (argv.length == 0)  
            hostName = "localhost";  
        else  
            hostName = argv[0];
```

```
try {
    Socket sock = new Socket(hostName, TIME_PORT);
    ObjectInputStream is = new ObjectInputStream(new BufferedInputStream(sock.getInputStream()));
    // Read and validate the Object
    Object o = is.readObject();
    if (o == null) {
        System.err.println("Read null from server!");
    } else if ((o instanceof Date)) {
        // Valid, so cast to Date, and print
        Date d = (Date) o;
        System.out.println("Server host is " + hostName);
        System.out.println("Time there is " + d.toString());
    } else {
        throw new IllegalArgumentException("Wanted Date, got " + o);
    }
}
```

```
} catch (ClassNotFoundException e) {
    System.err.println("Wanted date, got INVALID CLASS (" + e + ")");
} catch (IOException e) {
    System.err.println(e);
}
}
```

readObject() reads an object from the serialized class. This method is used to call the defaultReadObject.

The java instanceof operator is used to test whether the **object** is an instance of the specified type (**class** or subclass or interface).

Connection-less socket programming

Java **DatagramSocket** and **DatagramPacket** classes are used for connection-less socket programming.

1. Java DatagramSocket class:

- Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets.
- A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Commonly used Constructors of DatagramSocket class

DatagramSocket() throws SocketException: it creates a datagram socket and binds it with the available Port Number on the localhost machine.

DatagramSocket(int port) throws SocketException: it creates a datagram socket and binds it with the given Port Number.

DatagramSocket(int port, InetAddress address) throws SocketException: it creates a datagram socket and binds it with the specified port number and host address.

2. Java DatagramPacket class:

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Commonly used Constructors of DatagramPacket class:

DatagramPacket(byte[] barr, int length): it creates a datagram packet. This constructor is used to **receive** the packets.

DatagramPacket(byte[] barr, int length, InetAddress address, int port): it creates a datagram packet. This constructor is used to **send** the packets.

```
//Example of Sending DatagramPacket by DatagramSocket
```

```
import java.net.*;
public class DSender{
public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome java";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
}
```

The **getBytes()** method encodes a given String into a sequence of bytes and returns an array of bytes.

```
DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
```

```
ds.send(dp);
```

```
ds.close();
```

```
}
```

```
}
```

//Example of Receiving DatagramPacket by DatagramSocket

```
//DReceiver.java
import java.net.*;
public class DReceiver{
public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
}
```

getData() method of Java DatagramPacket class returns the data buffer. Any data that is to be received or is to be sent, firstly starts from the offset in the buffer and then runs for length long.

UDP Datagrams

Problem

You need to use a datagram connection (UDP) instead of a stream connection (TCP).

Solution

Use DatagramSocket and DatagramPacket.

Basic Steps: UDP Client

1. Create a DatagramSocket with no arguments (the form that takes two arguments is used on the server).
2. Optionally connect() the socket to an InetAddress and port number.
3. Create one or more DatagramPacket objects; these are wrappers around a byte array that contains data you want to send and is filled in with data you receive.
4. If you did not connect() the socket, provide the InetAddress and port when constructing the DatagramPacket.
5. Set the packet's length and use sock.send(packet) to send data to the server.
6. Use sock.receive() to retrieve data.

Difference between TCP and UDP

TRANSMISSION CONTROL PROTOCOL (TCP)	USER DATAGRAM PROTOCOL (UDP)
TCP is a connection-oriented protocol. Connection-orientation means that the communicating devices should establish a connection before transmitting data and should close the connection after transmitting the data.	UDP is the Datagram oriented protocol. This is because there is no overhead for opening a connection, maintaining a connection, and terminating a connection. UDP is efficient for broadcast and multicast type of network transmission.
TCP is reliable as it guarantees delivery of data to the destination router.	The delivery of data to the destination cannot be guaranteed in UDP.
TCP provides extensive error checking mechanisms. It is because it provides flow control and acknowledgment of data.	UDP has only the basic error checking mechanism using checksums.
Sequencing of data is a feature of Transmission Control Protocol (TCP). this means that packets arrive in-order at the receiver.	There is no sequencing of data in UDP. If ordering is required, it has to be managed by the application layer.
TCP is comparatively slower than UDP.	UDP is faster, simpler and more efficient than TCP.
Retransmission of lost packets is possible in TCP, but not in UDP.	There is no retransmission of lost packets in User Datagram Protocol (UDP).
TCP has a (20-80) bytes variable length header.	UDP has a 8 bytes fixed length header.
TCP is heavy-weight.	UDP is lightweight.
TCP doesn't supports Broadcasting.	UDP supports Broadcasting.
TCP is used by HTTP, HTTPS, FTP, SMTP and Telnet.	UDP is used by DNS, DHCP, TFTP, SNMP, RIP, and VoIP.

Example 13-8. DaytimeUDP.java

```
public class DaytimeUDP {  
    /** The UDP port number */  
    public final static int DAYTIME_PORT = 13;  
    /** A buffer plenty big enough for the date string */  
    protected final static int PACKET_SIZE = 100;  
    /** The main program that drives this network client.  
     * @param argv[0] hostname, running daytime/udp server  
     */  
    public static void main(String[] argv) throws IOException {  
        if (argv.length < 1) {  
            System.err.println("usage: java DayTimeUDP host");  
            System.exit(1);  
        }  
        String host = argv[0];  
        InetAddress servAddr = InetAddress.getByName(host);  
        DatagramSocket sock = new DatagramSocket();  
  
        //sock.connect(servAddr, DAYTIME_PORT);  
        byte[] buffer = new byte[PACKET_SIZE];
```

```
// The udp packet we will send and receive
DatagramPacket packet = new DatagramPacket(buffer, PACKET_SIZE, servAddr, DAYTIME_PORT);

/* Send empty max-length (-1 for null byte) packet to server */
packet.setLength(PACKET_SIZE-1);
sock.send(packet);
System.out.println("Sent request");

// Receive a packet and print it.
sock.receive(packet);
System.out.println("Got packet of size " + packet.getLength());
System.out.print("Date on " + host + " is " +
new String(buffer, 0, packet.getLength()));
sock.close();
}

}
```

Output:

```
Sent request
Got packet of size 26
Date on dalai is Sat Feb 8 20:22:12 2014
$
```

URI, URL, or URN

A URL is the traditional name for a network address consisting of a scheme (like “http”) and an address (site name) and resource or pathname. But there are three distinct terms in all:

- URI (Uniform Resource Identifier)
- URL (Uniform Resource Locator)
- URN (Uniform Resource Name)

Difference between URI, URL, and URN

The main difference between URI and URL is that every URL is a URI but not vice versa. Similarly, every URN is a URI, but the opposite is not true. Another difference between URI and URL is that URL includes the protocol, which is key to retrieving information from any location. Here are few differences between URI, URL and URN in point format:

- 1) Every URL and URN is URI because URI is the superset of both URL and URN.
- 2) URL includes protocol e.g. http://, ftp:// along with location to identify resource e.g. <http://www.blogspot.com/abc.html>.
- 3) URN are the unambiguous way to identify a resource.

URIDemo.java

```
public class URIDemo {  
    public static void main(String[] args)  
        throws URISyntaxException, MalformedURLException {  
        URI u = new URI("http://www.darwinsys.com/java/..//openbsd/..//index.jsp");  
        System.out.println("Raw: " + u);  
        URI normalized = u.normalize();  
        System.out.println("Normalized: " + normalized);  
        final URI BASE = new URI("http://www.darwinsys.com");  
        System.out.println("Relativized to " + BASE + ": " + BASE.relativize(u));  
        // A URL is a type of URI  
        URL url = new URL(normalized.toString());  
        System.out.println("URL: " + url);  
        // Junk  
        URI uri = new URI("bean:WonderBean");  
        System.out.println(uri);  
    }  
}
```

The **normalize()** method used to return a path from current path in which all redundant name elements are eliminated.

The precise definition of this method is implementation dependent and it derives a path that does not contain redundant name elements. In many file systems, the “.” and “..” are special names indicating the current directory and parent directory. In those cases all occurrences of “.” are considered redundant and If a “..” is preceded by a non-“..” name then both names are considered redundant.

The **relativize(Path other)** method of used to create a relative path between this path and a given path as a parameter.

For example, if this path is “/dir1/dir2” and the given path as a parameter is “/dir1/dir2/dir3/file1” then this method will construct a relative path “dir3/file1”. Where this path and the given path do not have a root component, then a relative path can be constructed.

Server-Side JAVA

Opening a Server Socket for Business

Problem

You need to write a socket-based server.

Solution

Create a ServerSocket for the given port number.

Discussion

- The ServerSocket represents the “other end” of a connection
- it waits patiently for clients to come along and connect to it.
- You construct a ServerSocket with just the port number. Because it doesn’t need to connect to another host, it doesn’t need a particular host’s address as the client socket constructor does.
- Your next step is to await client activity, which you do by calling accept(). This call blocks until a client connects to your server; at that point, the accept() returns to you a Socket object (not a ServerSocket) that is connected in both directions to the Socket object on the client

- *if you want to explicitly specify a local IP address to be bound (in case the computer has multiple IP addresses).*

```
import java.net.*;
import java.io.*;
public class ListenInside {
    /** The TCP port for the service. */
    public static final short PORT = 9999;
    /** The name of the network interface. */
    public static final String INSIDE_HOST = "acmewidgets-inside";
    /** The number of clients allowed to queue */
    public static final int BACKLOG = 10;
    public static void main(String[] argv) throws IOException {
        ServerSocket sock;
        Socket clientSock;
        try {
            sock = new ServerSocket(PORT, BACKLOG,
                InetAddress.getByName(INSIDE_HOST));
```

```
        while ((clientSock = sock.accept()) != null) {
            // Process it.
            process(clientSock);
        }
    } catch (IOException e) {
        System.err.println(e)
    }
}

/** Hold server's conversation with
 * one client. */
static void process(Socket s) throws
IOException {
    System.out.println("Connected
from " + INSIDE_HOST +
": " + s.getInetAddress());
    // The conversation would be here.
    s.close();
}
```

static InetAddress getByName(String host)

Determines the IP address of a host, given the host's name.

Code for a socket-based server

```
public class Listen {  
    /** The TCP port for the service. */  
    public static final short PORT = 9999;  
    public static void main(String[] argv) throws IOException {  
        ServerSocket sock;  
        Socket clientSock;  
        try {  
            sock = new ServerSocket(PORT);  
            while ((clientSock = sock.accept()) != null) {  
                // Process it.  
                process(clientSock);  
            }  
        } catch (IOException e) {  
            System.err.println(e);  
        }  
    }  
    /** This would do something with one client. */  
    static void process(Socket s) throws IOException {  
        System.out.println("Accept from client " +  
                           s.getInetAddress());  
        // The conversation would be here.  
        s.close();}  
    }  
}
```

Returning a Response (String)

Problem

- You need to write a string data to the client.

Solution

- The socket gives you an InputStream and an OutputStream. Use them.

Discussion

- It is possible to send data from the server and receive a response from the client. Similarly, the client can also send and receive data to-and-from.
- ECHO server gets a text from a client and sends it back to him. Such servers were frequently used for testing.

```
public class EchoServer {  
    /** Our server-side rendezvous socket */  
    protected ServerSocket sock;  
    /** The port number to use by default */  
    public final static int ECHOPORT = 7;  
    /** Flag to control debugging */  
    protected boolean debug = true;  
    /** main: construct and run */  
    public static void main(String[] args) {  
        int p = ECHOPORT;  
        if (args.length == 1) {  
            try {  
                p = Integer.parseInt(args[0]);  
            } catch (NumberFormatException e) {  
                System.err.println("Usage: EchoServer [port#]");  
                System.exit(1);  
            }  
        }  
        new EchoServer(p).handle();  
    }  
}
```

```
/** Construct an EchoServer on the given port number */  
public EchoServer(int port) {  
    try {  
        sock = new ServerSocket(port);  
    } catch (IOException e) {  
        System.err.println("I/O error in setup");  
        System.err.println(e);  
        System.exit(1);  
    }  
}  
/* This handles the connections */  
protected void handle() {  
    Socket ios = null;  
    BufferedReader is = null;  
    PrintWriter os = null;  
    while (true) {  
        try {  
            System.out.println("Waiting for client...");  
            ios = sock.accept();  
            System.err.println("Accepted from " +  
                ios.getInetAddress().getHostName());  
        }  
    }  
}
```

```
is = new BufferedReader(  
    new InputStreamReader(ios.getInputStream(), "8859_1"));  
os = new PrintWriter(new OutputStreamWriter(ios.getOutputStream(),  
    "8859_1"), true);  
String echoLine;  
while ((echoLine = is.readLine()) != null) {  
    System.err.println("Read " + echoLine);  
    os.print(echoLine + "\r\n");  
    os.flush();  
    System.err.println("Wrote " + echoLine);  
}  
System.err.println("All done!");  
}  
  
catch (IOException e) {  
    System.err.println(e);  
} finally {  
    try {  
        if (is != null)  
            is.close();  
        if (os != null)  
            os.close();  
        if (ios != null)  
            ios.close();  
    } catch (IOException e) {  
        // These are unlikely, but might indicate  
        // that  
        // the other end shut down early, a disk  
        // filled up  
        // but wasn't detected until close, etc.  
        System.err.println("IO Error in close");  
    }  
}  
}  
}  
/*NOTREACHED*/  
}
```

Returning Object Information Across a Network Connection

Problem

You need to return an object across a network connection.

Solution

Create the object you need, and write it using an `ObjectOutputStream` created on top of the socket's output stream.

Discussion

The `DaytimeObjectServer` (the other end of that process), is a program that constructs a `Date` object each time it's connected to and returns it to the client.

```
import java.io.ObjectOutputStream;
public class DaytimeObjectServer {
    /** The TCP port for the object time service. */
    public static final short TIME_PORT = 1951;
    public static void main(String[] argv) {
        ServerSocket sock;
        Socket clientSock;
        try {
            sock = new ServerSocket(TIME_PORT);
            while ((clientSock = sock.accept()) != null) {
                System.out.println("Accept from " + clientSock.getInetAddress());
                ObjectOutputStream os = new
                    ObjectOutputStream(clientSock.getOutputStream());
                // Construct and write the Object
                os.writeObject(new Date());
                os.close();
            }
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```

Handling Multiple Clients

Problem

Your server needs to handle multiple clients.

Solution

Use a thread for each.

Discussion

- Java uses the general-purpose Thread mechanism.
- Each time the code accepts a new connection from the ServerSocket, it immediately constructs and starts a new thread object to process that client.
- We want our architecture to support multiple clients at the same time. For this reason, we must use threads on server side so that whenever a client request comes, a separate thread can be assigned for handling each request.



Handling Multiple Clients(continue..)

Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the `java.lang.Thread` class.

There are two ways to create thread in java;

- Implement the `Runnable` interface (`java.lang.Runnable`)
- By Extending the `Thread` class (`java.lang.Thread`)

Create a Thread by Extending a Thread Class

The second way to create a thread is to create a new class that extends `Thread` class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in `Thread` class.

Step 1

You will need to override `run()` method available in `Thread` class. This method provides an entry point for the thread and you will put your complete business logic inside this method.

Step 2

Once `Thread` object is created, you can start it by calling `start()` method, which executes a call to `run()` method.



Handling Multiple Clients(continue..)

How these programs works together?

1. When a client, say client1 sends a request to connect to server, the server assigns a new thread to handle this request. The newly assigned thread is given the access to streams for communicating with the client.
2. After assigning the new thread, the server via its while loop, again comes into accepting state.
3. When a second request comes while first is still in process, the server accepts this requests and again assigns a new thread for processing it. In this way, multiple requests can be handled even when some requests are in process.



```

public class EchoServerThreaded2{
    public static final int ECHOPORT = 7;
    public static final int NUM_THREADS = 4;
    /** Main method, to start the servers. */
    public static void main(String[] av) {
        new EchoServerThreaded2(ECHOPORT, NUM_THREADS);
    }
    /** Constructor */
    public EchoServerThreaded2(int port, int numThreads) {
        ServerSocket servSock;
        try {
            servSock = new ServerSocket(port);
        } catch(IOException e) {
            /* Crash the server if IO fails. Something bad has happened */
            throw new RuntimeException("Could not create ServerSocket",
                e);
        }
        // Create a series of threads and start them.
        for (int i=0; i<numThreads; i++) {
            new Handler(servSock, i).start();
        }
    }
}

```

/** A Thread subclass to handle one client conversation.

```

*/
class Handler extends Thread {
    ServerSocket servSock;
    int threadNumber;
    /** Construct a Handler. */
    Handler(ServerSocket s, int i) {
        servSock = s;
        threadNumber = i;
        setName("Thread " + threadNumber);
    }
}

```



```
public void run() {  
/* Wait for a connection. Synchronized on the ServerSocket  
* while calling its accept() method.  
*/  
while (true) {  
try {  
System.out.println(getName() + " waiting");  
Socket clientSocket;  
// Wait here for the next connection.  
synchronized(servSock){  
clientSocket = servSock.accept();  
}  
System.out.println(getName() + " starting, IP=" +clientSocket.getInetAddress());  
BufferedReader is = new BufferedReader(  
new InputStreamReader(clientSocket.getInputStream()));  
PrintStream os = new PrintStream(clientSocket.getOutputStream(), true);  
String line;  
while ((line = is.readLine()) != null) {  
os.print(line + "\r\n");  
os.flush();  
}  
System.out.println(getName() + " ENDED ");  
clientSocket.close();  
} catch (IOException ex) {  
System.out.println(getName() + ": IO Error on socket " + ex);  
return;  
}  
}  
}  
}  
}
```



Returning Object Information Across a Network Connection

Problem

You need to return an object across a network connection.

Solution

Create the object you need, and write it using an `ObjectOutputStream` created on top of the socket's output stream.

Discussion

The `DaytimeObjectServer` (the other end of that process), is a program that constructs a `Date` object each time it's connected to and returns it to the client.



```
import java.io.ObjectOutputStream;
public class DaytimeObjectServer1 {
    /** The TCP port for the object time service. */
    public static final short TIME_PORT = 1951;
    public static void main(String[] argv) {
        ServerSocket sock;
        Socket clientSock;
        try {
            sock = new ServerSocket(TIME_PORT);
            while ((clientSock = sock.accept()) != null) {
                System.out.println("Accept from " + clientSock.getInetAddress());
                ObjectOutputStream os = new ObjectOutputStream(clientSock.getOutputStream());
                // Construct and write the Object
                os.writeObject(new Date());
                os.close();
            }
        } catch (IOException e) {
            System.err.println(e);
        }
    }
}
```



Handling Multiple Clients

Problem

Your server needs to handle multiple clients.

Solution

Use a thread for each.

Discussion

- Java uses the general-purpose Thread mechanism.
- Each time the code accepts a new connection from the ServerSocket, it immediately constructs and starts a new thread object to process that client.
- We want our architecture to support multiple clients at the same time. For this reason, we must use threads on server side so that whenever a client request comes, a separate thread can be assigned for handling each request.



Handling Multiple Clients(continue..)

Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the `java.lang.Thread` class.

There are two ways to create thread in java;

- Implement the `Runnable` interface (`java.lang.Runnable`)
- By Extending the `Thread` class (`java.lang.Thread`)

Create a Thread by Extending a Thread Class

The second way to create a thread is to create a new class that extends `Thread` class using the following two simple steps. This approach provides more flexibility in handling multiple threads created using available methods in `Thread` class.

Step 1

You will need to override `run()` method available in `Thread` class. This method provides an entry point for the thread and you will put your complete business logic inside this method.

Step 2

Once `Thread` object is created, you can start it by calling `start()` method, which executes a call to `run()` method.



Handling Multiple Clients(continue..)

How these programs works together?

1. When a client, say client1 sends a request to connect to server, the server assigns a new thread to handle this request. The newly assigned thread is given the access to streams for communicating with the client.
2. After assigning the new thread, the server via its while loop, again comes into accepting state.
3. When a second request comes while first is still in process, the server accepts this requests and again assigns a new thread for processing it. In this way, multiple requests can be handled even when some requests are in process.



```

public class EchoServerThreaded{
    public static final int ECHOPORT = 7;
    public static final int NUM_THREADS = 4;
    /** Main method, to start the servers. */
    public static void main(String[] av) {
        new EchoServerThreaded2(ECHOPORT, NUM_THREADS);
    }
    /** Constructor */
    public EchoServerThreaded2(int port, int numThreads) {
        ServerSocket servSock;
        try {
            servSock = new ServerSocket(port);
        } catch(IOException e) {
            /* Crash the server if IO fails. Something bad has happened */
            throw new RuntimeException("Could not create ServerSocket",
                e);
        }
        // Create a series of threads and start them.
        for (int i=0; i<numThreads; i++) {
            new Handler(servSock, i).start();
        }
    }
}

```

/** A Thread subclass to handle one client conversation.

```

*/
class Handler extends Thread {
    ServerSocket servSock;
    int threadNumber;
    /** Construct a Handler. */
    Handler(ServerSocket s, int i) {
        servSock = s;
        threadNumber = i;
        setName("Thread " + threadNumber);
    }
}

```



```
public void run() {  
/* Wait for a connection. Synchronized on the ServerSocket  
* while calling its accept() method.  
*/  
while (true) {  
try {  
System.out.println(getName() + " waiting");  
Socket clientSocket;  
// Wait here for the next connection.  
synchronized(servSock){  
clientSocket = servSock.accept();  
}  
System.out.println(getName() + " starting, IP=" +clientSocket.getInetAddress());  
BufferedReader is = new BufferedReader(  
new InputStreamReader(clientSocket.getInputStream()));  
PrintStream os = new PrintStream(clientSocket.getOutputStream(), true);  
String line;  
while ((line = is.readLine()) != null) {  
os.print(line + "\r\n");  
os.flush();  
}  
System.out.println(getName() + " ENDED ");  
clientSocket.close();  
} catch (IOException ex) {  
System.out.println(getName() + ": IO Error on socket " + ex);  
return;  
}  
}  
}  
}  
}
```



Serving the HTTP Protocol

HTTP means *HyperText Transfer Protocol*. HTTP is the underlying protocol used by the World Wide Web and this protocol defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

For example, when you enter a URL in your browser, this actually sends an HTTP command to the Web server directing it to fetch and transmit the requested Web page.

Problem

You want to serve up a protocol such as HTTP.

Solution

Create a ServerSocket and write some code that “speaks” the particular protocol. Or, better, use a Java-powered web server such as Apache Tomcat or a Java Enterprise Edition(Java EE) server such as JBoss WildFly.

Discussion

You can implement your own HTTP protocol server for very simple applications, which we'll do here.

How to make HTTP Server in Java

First step to create a web server is, to create a network socket which can accept connection on certain TCP port. HTTP server usually listen on port 80 but we will use a different port 8080 for testing purpose. You can use ServerSocket class in Java to create a Server which can accept requests, as shown in the next slide.



```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.ServerSocket;
import java.net.Socket;

public class SimpleHTTPServer {
    public static void main(String args[] ) throws IOException {
        ServerSocket server = new ServerSocket(8080);
        System.out.println("Listening for connection on port 8080 ....");
        while (true) {
            Socket clientSocket = server.accept();
            InputStreamReader isr = new InputStreamReader(clientSocket.getInputStream());
            BufferedReader reader = new BufferedReader(isr);
            String line = reader.readLine();
            while (!line.isEmpty()) {
                System.out.println(line);
                line = reader.readLine();
            }
        }
    }
}
```



When you connect to this server using Firefox it will spin endlessly but on server side you will see following lines on your console :

Listening for connection on port 8080

GET / HTTP/1.1

Host: localhost:8080

.

.

.

Connection: Keep-alive

So now our server is not only listening for connection, but accepting it and also reading HTTP request. Now only thing remaining is to send HTTP response back to the client. To keep our server simple, we will just send today's date to the client. Let's see how we can do that. In order to send response, we need to get the output stream from socket and then we will write HTTP response code OK and today's date into stream.



```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Date;

/*Java program to create a simple HTTP Server to demonstrate how to use ServerSocket and Socket class.*/
public class SimpleHTTPServer {
    public static void main(String args[]) throws IOException { ServerSocket server = new ServerSocket(8080);
        System.out.println("Listening for connection on port 8080 ....");
        while (true) {
            try (Socket socket = server.accept())
            {
                Date today = new Date();
                String httpResponse = "HTTP/1.1 200 OK\r\n\r\n" + today;
                socket.getOutputStream().write(httpResponse.getBytes("UTF-8"));
            }
        }
    }
}
```

When you run the above program in Eclipse or from command line and connect to the <http://localhost:8080> from Firefox, you will see following response :

Thurs Apr 22 13:32:26 GMT+08:00 2020

Which is today's date. It means *our HTTP Server is working properly*, it is listening on port 8080, accepting connection, reading request and sending response.



Only limitation of this server is that it can serve one client at a time. If request processing takes longer time, the other connection has to wait. This problem can be solved by using threads or Java NIO non blocking selectors and channels.



Chapter 18

Database Access

18.1 Introduction

- ☞ Java Can be used to access many kind of database.
- ☞ A database can be a text file, a fast key/value pairing on disk (DBM format), a SQL-based relational database management system(DBMS), or an object database.
- ☞ This chapter focuses on relational database.

18.1.1 JDBC(Java Database Connectivity)

- ☞ JDBC is a Java API to connect and execute the query with the database.
- ☞ It is a part of JavaSE(Java Standard Edition).
- ☞ JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers.
 - * JDBC-ODBC Bridge Driver
 - * Native Driver,
 - * Network Protocol Driver, and
 - * Thin Driver
- ☞ We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database.
- ☞ It is like Open Database Connectivity (ODBC) provided by Microsoft.
- ☞ To access the database, we need to import java.sql package.
- ☞ This contains classes and interfaces for JDBC API. Some of the popular interfaces of JDBC API are as follows.
 - * Driver interface

- * Connection interface
 - * Statement interface
 - * PreparedStatement interface
 - * CallableStatement interface
 - * ResultSet interface
 - * ResultSetMetaData interface
 - * DatabaseMetaData interface
 - * RowSet interface
- ☞ Popular classes of JDBC API:
- * DriverManager class
 - * Blob class
 - * Clob class
 - * Types class

18.1.2 Why Should We Use JDBC?

- ☞ Before JDBC, ODBC API was the database API to connect and execute the query with the database.
- ☞ ODBC API uses ODBC driver, which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).
- ☞ Hence, JDBC API can be used to handle database using Java program and can perform the following activities.
 - * Connect to the database
 - * Execute queries and update statements to the database
 - * Retrieve the result received from the database.

18.1.3 JDBC Drivers

18.1.3.1 JDBC-ODBC bridge driver

- ☞ The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

Advantages:

- ☞ Easy to use.
- ☞ Can be easily connected to any database.

Disadvantages:

- ☞ Performance degraded because JDBC method call is converted into the ODBC function calls.
- ☞ The ODBC driver needs to be installed on the client machine.

Note: Oracle does not support the JDBC-ODBC Bridge from Java 8.

18.1.3.2 Native-API driver

- ☞ The Native API driver uses the client-side libraries of the database.
- ☞ The driver converts JDBC method calls into native calls of the database API.
- ☞ It is not written entirely in java.

Advantage:

- ☞ performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- ☞ The Native driver needs to be installed on the each client machine.
- ☞ The Vendor client library needs to be installed on client machine.

18.1.3.3 Network Protocol driver

- ☞ The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol.
- ☞ It is fully written in java.

Advantages:

- ☞ No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- ☞ Network support is required on client machine.
- ☞ Requires database-specific coding to be done in the middle tier.
- ☞ Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

18.1.3.4 Thin Driver

- ☞ The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

Advantages:

- ☞ Better performance than all other drivers.
- ☞ No software is required at client side or server side.

Disadvantages:

- ☞ Drivers depend on the Database.

18.1.4 JDBC Setup and Connection

- ☞ To access a database via JDBC, use `Class.forName()` and `DriverManager.getConnection()`

Prerequisite:

- ☞ Some knowledge about SQL, the universal language used to control relational database.

Example:

```
"SELECT * from table_name";
```

*—> All columns to be selected from all rows as SELECT statement has no where clause.

Steps to connect:

- ☞ To connect any java application with the database using JDBC, we require 5 steps.
 - * Register the Driver class
 - * Create connection (Section 18.3 in the book)
 - * Create statement (Section 18.4 in the book)
 - * Execute queries (Section 18.4 in the book)
 - * Close connection

18.1.4.1 Register the Driver class

- ☞ In this section, we need to register the driver class.
- ☞ To do so, `forName()` method of `Class` class is used. This function send the class name as parameter.
- ☞ `Class.forName` will automatically register the `DriverManager` class.

Syntax (prototype)

```
public static void forName(String class name) throws ClassNotFoundException
```

- ☞ So, JDBC-ODBC bridge driver can be loaded as follows.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver"); //sun: the driver should  
be present in Sun JDK implementation
```

- ☞ Similarly, oracle Driver can be loaded as follows.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

18.1.4.2 Connecting to a JDBC database

JDBC URL

- ☞ URL represents a protocol to connect to a database.
- ☞ JDBC driver uses a JDBC URL to identify and connect to a particular database.
- ☞ This URL follows a common pattern:

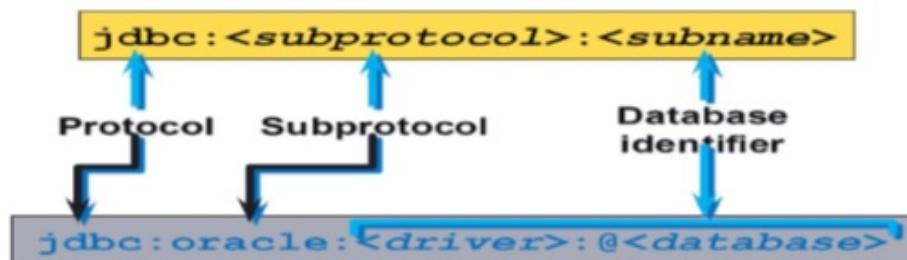


Figure 18.1: Pattern for JDBC URL

- ☞ Different database require different information to connect successfully.

Example

- * For oracle JDBC-Thin driver
`jdbc:oracle:thin:@site:port:database`
- * For JDBC-ODBC Bridge Driver
`jdbc:odbc:datasource:odbcoptions`

- ☞ The only requirement is that a driver be able to recognize its own URLs.

URL Requirements

1. Host Details

- ☞ This is the IP address of the system or the host name from where Oracle database is being accessed.
- ☞ If your Oracle database is running on the same system, where you are executing your jdbc program then you can use @ localhost (in case of the IP address) in your JDBC URL as follows.

```
jdbc:oracle:thin:@localhost
```

- ☞ If it is different system, the IP address of that host need to be found out. Now that IP can be included in your JDBC URL as follows.

```
jdbc.oracle.thin@192.168.10.5
```

2. Service Name and Port Number

- ☞ The service name and port number on which Oracle service is running is a requirement in the JDBC URL. You may get all these information from the tnsnames.ora file.
- ☞ Let in our case the service name and port number are orcl and 1521 respectively. Now after adding service name and port number, the URL can be as follows.

```
jdbc:oracle:thin:@192:16810.5:1521:orcl
```

- ☞ The connection URL for the oracle database is

```
jdbc:oracle:thin:@localhost:1521:orcl,
```

where jdbc is the API/ protocol, oracle is the database/ sub protocol, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and orcl is the Oracle service name.

3. Create the connection object

- ☞ To create a connection object using this JDBC URL, the `getConnection()` method of `DriverManager` class is used to establish the connection with the database.

Syntax

- * Database that does not require explicit login.

```
public static Connection getConnection(String URL) throws SQLException
```

- * Database that requires explicit login.

```
public static Connection getConnection(String URL, String username,  
String password)
```

Username: The default username for the oracle database is system.

Password: It is the password given by the user at the time of installing the oracle database.

- ☞ When the method is called, the `DriverManager` queries each registered driver, asking if it understands the URL. If a driver recognizes the URL, it returns a `Connection` object.

- ☞ Using all above, the complete URL can be formed as follows.

```
Connection con=DriverManger.getConnection("jdbc:oracle:thin:@localhost:1521:orcl","SYSTEM", "subrat")
```

Assignment

1. Write a program to establish a connection to the Oracle database.
2. Write a program that displays successful connection message on establishment of a connection to the Oracle database and throws exception otherwise.

18.1.4.3 Sending a JDBC query and getting results

Create a Statement object

- ☞ Once a connection is established, it is used to pass SQL statements to its undergoing database.
- ☞ A Statement object is used to send SQL statements to a database.
- ☞ The Statement interface provides basic methods for executing statements and retrieving results.
- ☞ Hence, to execute SQL statements, we can use Statement object. Before this, we need an active connection to create a JDBC statement.
- ☞ The `createStatement()` method of Connection interface is used to create Statement object.

Syntax:

```
public Statement createStatement() throws SQLException
```

Example:

```
Statement stmt=con.createStatement()
```

Execute the Query

- ☞ The `executeQuery()` method of Statement interface is used to execute queries to the database.
- ☞ This method returns the object of ResultSet interface that can be used to get all the records of a table.
- ☞ The ResultSet works like an iterator in that it lets you access all the rows of the result that match the query

Syntax: `public ResultSet executeQuery(String sql) throws SQLException`

Example: `ResultSet rs=stmt.executeQuery("SELECT * from table_name")`

- ☞ ResultSet is an object that contains the results of executing a SQL statement. Use `next()` to step through the result set row by row.
- ☞ To retrieve the data from the columns, we can use `getXXX()` method, e.g., `rs.getInt()`, `rs.getString()`

JDBC method	SQL type	Java type	JDBC method	SQL type	Java type
getBit()	BIT	boolean	getDate()	DATE	java.sql.Date
getByte()	TINYINT	byte	getTimeStamp()	TIME	java.sql.Date
getShort()	SMALLINT	short	getObject()	BLOB	Object
getInt()	INTEGER	int			
getLong()	BIGINT	long			
getFloat()	REAL	float			
getDouble()	DOUBLE	double			
getString()	CHAR	String			
getString()	VARCHAR	String			
getString()	LONGVARCHAR	String			

Figure 18.2: Data type mappings between SQL and JDBC

Assignment

1. Write the code to extract entire employee details from an existing table in the Oracle database.
2. Write the code that displays the name of the employee and employee id, whose salary is above Rs 50000 from an existing employee table in the Oracle database.

Example:

```

static final String JDBC_DRIVER =
    "oracle.jdbc.driver.OracleDriver";
static final String DB_URL =
    "jdbc:oracle:thin:@localhost:1521:orcl";
// Database credentials
static final String USER = "hr";
static final String PASS = "subrat";
//STEP 2: Register JDBC driver
Class.forName("oracle.jdbc.driver.OracleDriver");
//STEP 3: Open a connection
System.out.println("Connecting to database...");
conn = DriverManager.getConnection(DB_URL,USER,PASS);

//STEP 4: Execute a query
System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT t_id, t_firstname, t_lastname FROM table1";
ResultSet rs = stmt.executeQuery(sql);
//STEP 5: Extract data from result set
while(rs.next()){
    //Retrieve by column name
    int id   = rs.getInt("t_id");
    String first = rs.getString("t_firstname");
    String last = rs.getString("t_lastname");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
}

```

18.1.5 Using JDBC Prepared Statements

- ☞ An SQL query consists of textual characters. The database must first parse a query and then compile it into something that can be run in the database. This can add up to a lot of overhead if you are sending a lot of queries.
- ☞ In some types of applications, you'll use a number of queries that are the same syntactically but have different values:

```
select * from payroll where personnelNo = 12345;
select * from payroll where personnelNo = 23740;
select * from payroll where personnelNo = 97120;
```

- ☞ In this case, the statement needs to be parsed and compiled only once. But if you keep making up select statements and sending them, the database mindlessly keeps parsing and compiling them.
- ☞ Better to use a prepared statement, in which the variable part is replaced by a parameter marker (a question mark). Then the statement need only be parsed (or organized, optimized, compiled, or whatever) once.

```
PreparedStatement ps = conn.prepareStatement("select * from payroll
where personnelNo = ?;")
```

- ☞ Before you can use this prepared statement, you must fill in the blanks with the appropriate set methods. These take a column number and the value to be plugged in. Then use `executeQuery()` with no arguments because the query is already stored in the statement.

```
ps.setInt(1, 12345)
rs = ps.executeQuery()
```

Example

Without Prepared Statements

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:
@localhost:1521:orcl","hr","subrat");
Statement stmt=con.createStatement();
rs=stmt.executeQuery("select * from table1 where
t_id=333");
//rs=stmt.executeQuery("select * from table1 where
t_id=111");
//rs=stmt.executeQuery("select * from table1 where
t_id=444");
rs.next();
System.out.println("T_Id: "+rs.getInt(1)+"\n"+"First
Name:
"+rs.getString(2)+"\n"+"Last Name:
"+rs.getString(3));
con.close();
```

With Prepared Statements

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe");
PreparedStatement ps=con.prepareStatement("select *
from table1 where t_id=?");
ps.setInt(1, 333);
//ps.setInt(1, 111);
//ps.setInt(1, 444);
rs=ps.executeQuery();
rs.next();
System.out.println("T_Id: "+rs.getInt(1)+"\n"+ "First
Name:"+rs.getString(2)+"\n"+ "Last Name:
"+rs.getString(3));

con.close();
```

Assignment

- Q. Write the code to extract entire employee details from an existing table in the Oracle database using preparedStatement. Show the advantages of using preparedStatement while extracting information.

18.1.6 Changing Data Using a ResultSet

- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row. However, ResultSet object can be moved forward only and it is not updatable by default. But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in `createStatement(int, int)` method as well as we can make this object as updatable by:

```
Statement stmt = con.createStatement(resultSet.TYPE_SCROLL_INSENSITIVE,
resultSet.CONCUR_UPDATABLE);
```

Common Methods:

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Figure 18.3: Common methods of ResultSet

Example

```

Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:orcl","hr","subrat
    ");
Statement stmt=con.createStatement(
    ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
ResultSet rs=stmt.executeQuery(" select * from
    table1");
//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "
    "+rs.getString(3));
rs.previous();
System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "
    "+rs.getString(3));
...
...
//so on
con.close();

```

18.1.7 Storing Results in a RowSet

- ☞ The instance of RowSet is the java bean component because it has properties and java bean notification mechanism.
- ☞ It is the wrapper of ResultSet. It holds tabular data like ResultSet but it is easy and flexible to use.
- ☞ The implementation classes of RowSet interface are as follows:
 - JdbcRowSet

- CachedRowSet
 - WebRowSet
 - JoinRowSet
 - FilteredRowSet
- ☞ The advantages of using RowSet are given below:
1. It is easy and flexible to use
 2. It is Scrollable and Updatable by default

Example:

Example

```
Class.forName("oracle.jdbc.driver.OracleDriver");
//Creating and Executing RowSet
JdbcRowSet rowSet =
    RowSetProvider.newFactory().createJdbcRowSet();
rowSet.setUrl("jdbc:oracle:thin:@localhost:1521:orcl");
rowSet.setUsername("hr");
rowSet.setPassword("subrat");
rowSet.setCommand("select * from table1");
rowSet.execute();
while (rowSet.next()) {
    // Generating cursor Moved event
    System.out.println("Id: " + rowSet.getInt(1));
    System.out.println("First Name: " +
        rowSet.getString(2));
    System.out.println("Second Name: " +
        rowSet.getString(3));
}
```

18.1.8 Changing Data Using SQL

- ☞ To insert or update data, create a new table, delete a table, or otherwise change the database, Instead of using the Statement method `executeQuery()`, use `executeUpdate()` with SQL commands to make the change.

Discussion:

- ☞ `boolean execute(String SQL)` : Returns a boolean value of true if a ResultSet object can be retrieved; otherwise, it returns false. Use this method to execute SQL DDL statements or when you need to use the truly dynamic SQL.
- ☞ `int executeUpdate(String SQL)`: Returns the number of rows affected by the execution of the SQL statement. Use this method to execute SQL statements, for which you expect to get a number of rows affected - for example, an INSERT, UPDATE, or DELETE statement.
- ☞ `ResultSet executeQuery(String SQL)`: Returns a ResultSet object. Use this method when you expect to get a result set, as you would with a SELECT statement.

Example

```
System.out.println("Creating statement...");  
stmt = conn.createStatement();  
String sql = "DELETE FROM TABLE1 WHERE  
    T_FIRSTNAME='Subrat';  
// String comparison  
  
// Let us check if it returns a true Result Set or not.  
Boolean ret = stmt.execute(sql);  
System.out.println("Return value is : " + ret.toString() );  
// SELECT returns a true Result Set  
  
// Let us update First Name of the record with ID = 444;  
sql = "DELETE FROM TABLE1 WHERE T_FIRSTNAME='Nikita';  
// or T_ID=444;  
int rows = stmt.executeUpdate(sql);  
System.out.println("Rows impacted : " + rows );
```

Assignment

- Q. Write the code to delete specific employee details from an existing table in the Oracle database using SQL command and display the number of rows affected by this process.

18.1.9 Finding JDBC Metadata

- ☞ To learn about a database or table, read the documentation provided by your vendor or database administrator. Or ask the software for a MetaData object.
- ☞ There are two classes relating to metadata (data about data) that you can ask for in the JDBC API: DatabaseMetaData and ResultSetMetaData.

ResultSetMetaData:

- ☞ we've been using a `getXXX()` method to retrieve a column value, knowing ahead of time the data type that was appropriate for a corresponding database column. But what if you didn't know? How would your program know how many columns are in the result set and what their data types are?
- ☞ To answer this question, you can use the methods provided by the ResultSetMetaData object.
- ☞ After you execute a SELECT statement and retrieve the ResultSet object, you can use the ResultSet object's `getMetaData()` method to retrieve a ResultSetMetaData object that will give you all the details you need to know to dynamically manipulate the ResultSet. The `getMetaData()` method has the following signature:
`ResultSetMetaData getMetaData()`

- ☞ An object that can be used to get information about the types and properties of the columns in a ResultSet object. The following code fragment creates the ResultSet object rs, creates the ResultSetMetaData object rsmd, and uses rsmd to find out how many columns rs has and whether the first column in rs can be used in a WHERE clause.

Example

```
ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM
    TABLE2");
ResultSetMetaData rsmd = rs.getMetaData();
    // creates the ResultSetMetaData
    // object
int numberOfColumns = rsmd.getColumnCount();
boolean b = rsmd.isSearchable(1);
// whether the first column in rs can be used in a WHERE
clause.
```

Some Methods:

- ✓ `int getColumnCount()` Returns the number of columns in this ResultSet object.
- ✓ `String getColumnName(int column)` Get the designated column's name.
- ✓ `int getColumnType(int column)` Retrieves the designated column's SQL type.
- ✓ `String getTableName(int column)` Gets the designated column's table name.
- ✓ `boolean isSearchable(int column)` Indicates whether the designated column can be used in a where clause.