Dr.SNS Rajalakshmi College of Arts and Science
(Autonomous)

Coimbatore-49

Department of Computer Applications

Oops- Unit –IV Notes

JAVA GUI, FILE STREAM AND CONCURRENCY

Prepared By

Dr.A.Devi

UNIT-IV

JAVA GUI, FILE STREAM AND CONCURRENCY

- Swing in Java is a Graphical User Interface (GUI) toolkit that includes the GUI components. Swing provides a rich set of widgets and packages to make sophisticated GUI components for Java applications.
- Swing is a part of Java Foundation Classes(JFC), which is an API for Java GUI programming that provide GUI.
- The Java Swing library is built on top of the Java Abstract Widget Toolkit (AWT), an older, platform dependent GUI toolkit.
- Java simple GUI programming components like button, textbox, etc., from the library and do not have to create the components from scratch.
- Java Swing is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
- Unlike AWT, Java Swing provides platform-independent and lightweight components.
- The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

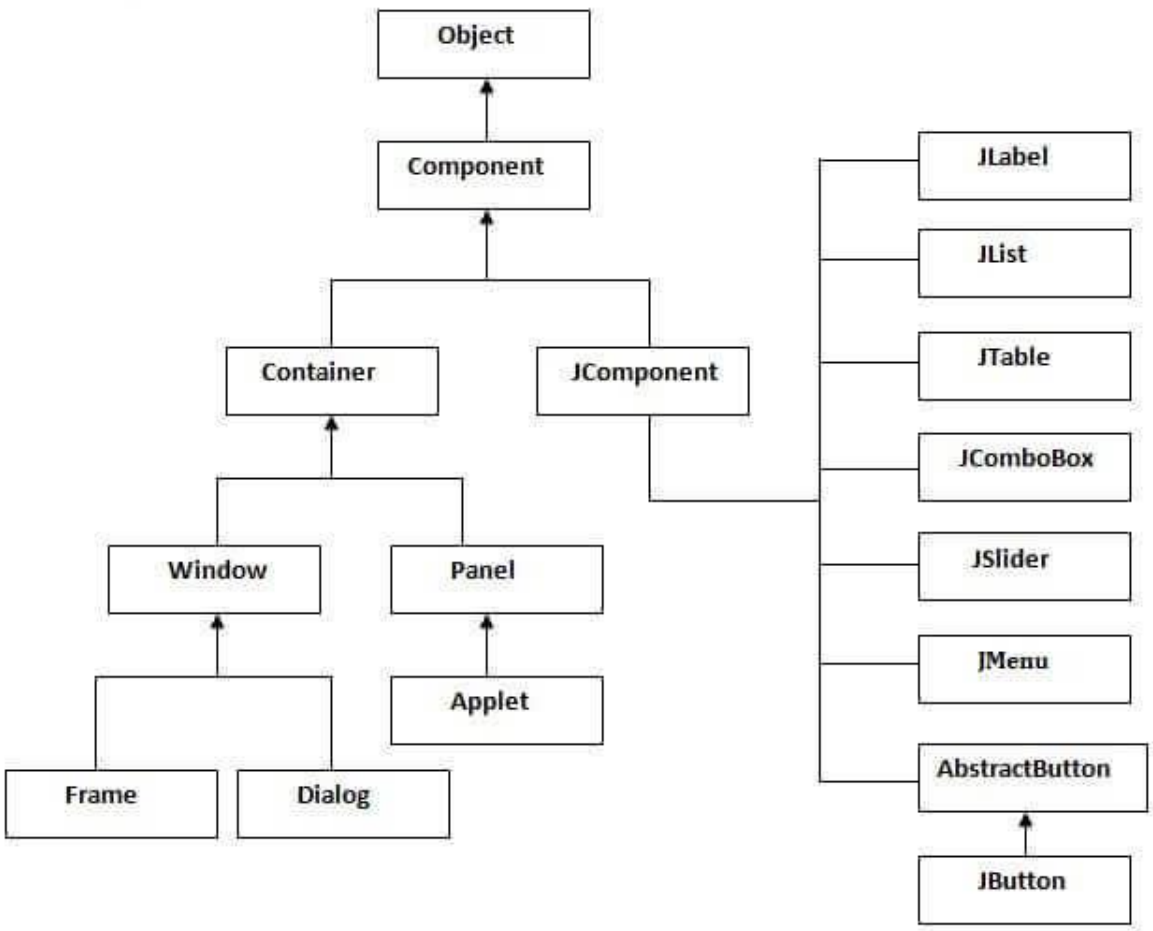
Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent.	Java swing components are platform-independent.
2)	AWT components are heavyweight.	Swing components are lightweight.
3)	AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follows MVC(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

JAVA SWING EXAMPLES

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

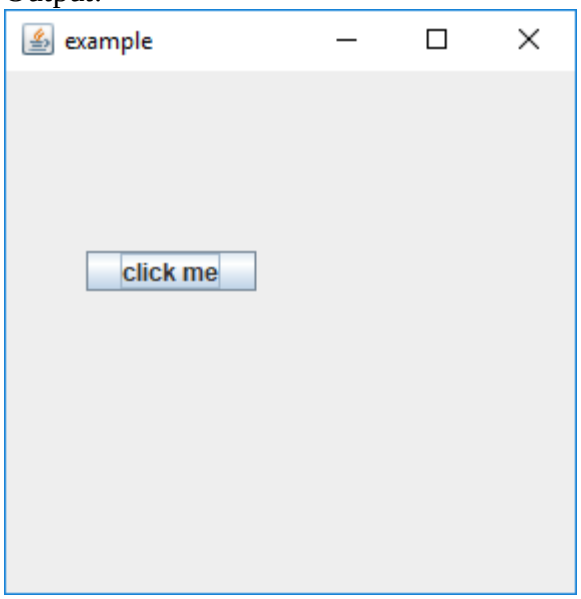
JButton Class

It is used to create a labelled button. Using the ActionListener it will result in some action when the button is pushed. It inherits the AbstractButton class and is platform independent.

Example:

```
 1import javax.swing.*;
 2public class example{
 3public static void main(String args[]) {
 4JFrame a = new JFrame("example");
 5JButton b = new JButton("click me");
 6b.setBounds(40,90,85,20);
 7a.add(b);
 8a.setSize(300,300);
 9a.setLayout(null);
10a.setVisible(true);
11}
12}
```

Output:



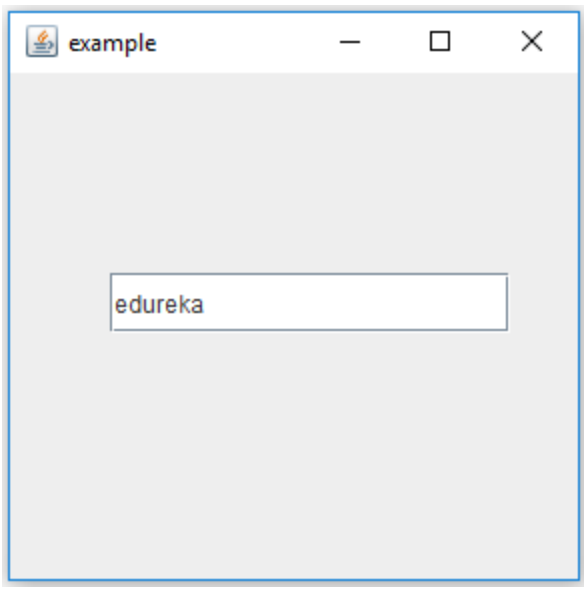
JTextFie

It inherits the JTextComponent class and it is used to allow editing of single line text.

Example:

```
import javax.swing.*;

public class example{

public static void main(String args[]) {

JFrame a = new JFrame("example");

JTextField b = new JTextField("edureka");

b.setBounds(50,100,200,30);

a.add(b);

a.setSize(300,300);

a.setLayout(null);

a.setVisible(true);

}

}
```

Output



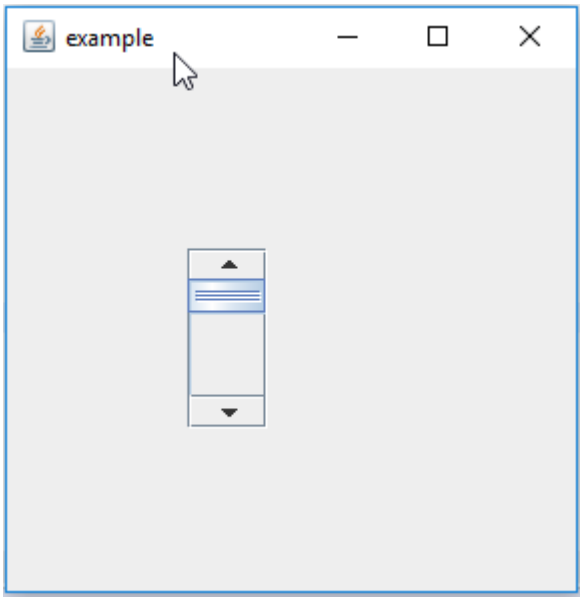
JSCROLLBAR CLASS

It is used to add scroll bar, both horizontal and vertical.

Example:

```
import javax.swing.*;
class example{
example(){
JFrame a = new JFrame("example");
JScrollBar b = new JScrollBar();
b.setBounds(90,90,40,90);
a.add(b);
a.setSize(300,300);
a.setLayout(null);
a.setVisible(true);
}
public static void main(String args[]){
new example();
}
}
```

Output

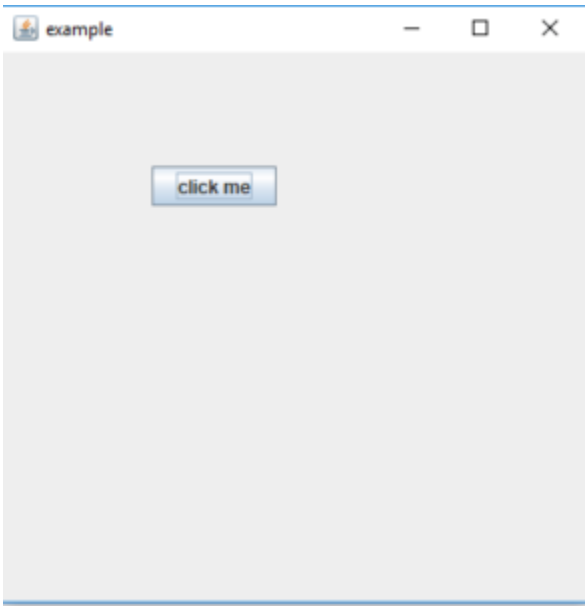


JPANEL CLASS

It inherits the JComponent class and provides space for an application which can attach any other component.

```
import java.awt.*;
```

```
import javax.swing.*;
public class Example{
Example(){
JFrame a = new JFrame("example");
JPanel p = new JPanel();
p.setBounds(40,70,200,200);
JButton b = new JButton("click me");
b.setBounds(60,50,80,40);
p.add(b);
a.add(p);
a.setSize(400,400);
a.setLayout(null);
a.setVisible(true);
}
public static void main(String args[])
{
new Example();
}
}
```

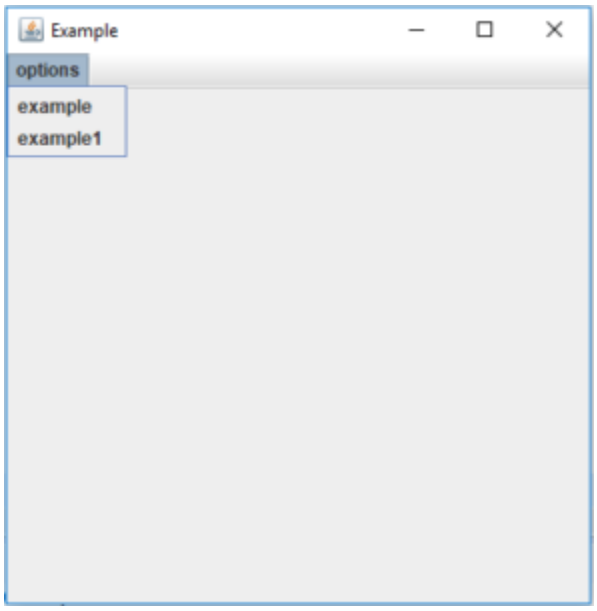


JMenu Class

It inherits the JMenuItem class, and is a pull down menu component which is displayed from the menu bar.

```
import javax.swing.*;
class Example{
JMenu menu;
JMenuItem a1,a2;
Example()
{
JFrame a = new JFrame("Example");
menu = new JMenu("options");
JMenuBar m1 = new JMenuBar();
a1 = new JMenuItem("example");
a2 = new JMenuItem("example1");
menu.add(a1);
menu.add(a2);
m1.add(menu);
a.setJMenuBar(m1);
a.setSize(400,400);
a.setLayout(null);
a.setVisible(true);
}
public static void main(String args[])
{
new Example();
}
}import javax.swing.*;
public class Example
{
Example(){
JFrame a  = new JFrame("example");
DefaultListModel<String> l = new DefaultListModel< >();
```

```
l.addElement("first item");
l.addElement("second item");
JList<String> b = new JList< >(l);
b.setBounds(100,100,75,75);
a.add(b);
a.setSize(400,400);
a.setVisible(true);
a.setLayout(null);
}
public static void main(String args[])
{
new Example();
}
}
```

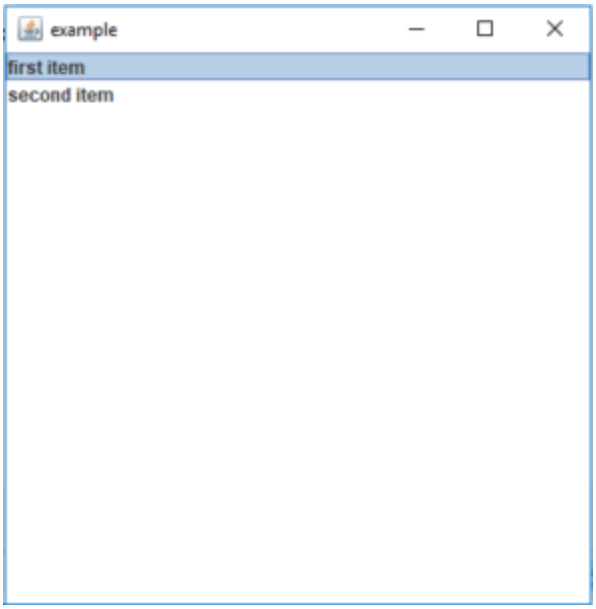


JLIST CLASS

It inherits JComponent class, the object of JList class represents a list of text items.

```
import javax.swing.*;
public class Example
{
Example(){
```

```
JFrame a  = new JFrame("example");
DefaultListModel<String> l = new DefaultListModel< >();
l.addElement("first item");
l.addElement("second item");
JList<String> b = new JList< >(l);
b.setBounds(100,100,75,75);
a.add(b);
a.setSize(400,400);
a.setVisible(true);
a.setLayout(null);
}
public static void main(String args[])
{
new Example();
}
}
```

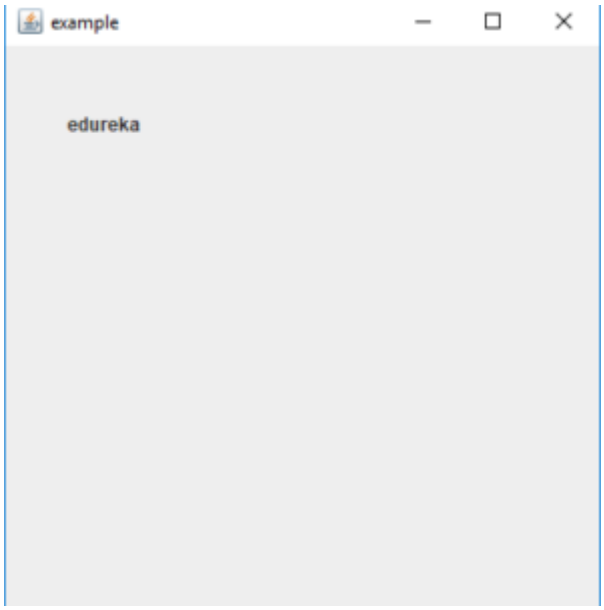


JLABEL CLASS

It is used for placing text in a container. It also inherits JComponent class.

```
import javax.swing.*;
public class Example{
```

```
public static void main(String args[])
{
JFrame a = new JFrame("example");
JLabel b1;
b1 = new JLabel("edureka");
b1.setBounds(40,40,90,20);
a.add(b1);
a.setSize(400,400);
a.setLayout(null);
a.setVisible(true);
}
}
```

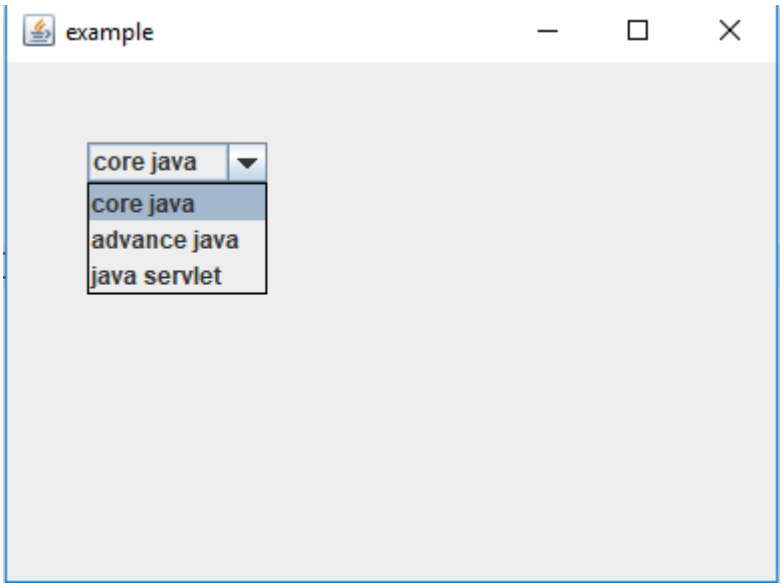


JCOMBOBOX CLASS

It inherits the JComponent class and is used to show pop up menu of choices.

```
import javax.swing.*;
public class Example{
JFrame a;
Example(){
a = new JFrame("example");
string courses[] = { "core java","advance java", "java servlet"};
```

```
JComboBox c = new JComboBox(courses);
c.setBounds(40,40,90,20);
a.add(c);
a.setSize(400,400);
a.setLayout(null);
a.setVisible(true);
}
public static void main(String args[])
{
new Example();
}
}
```



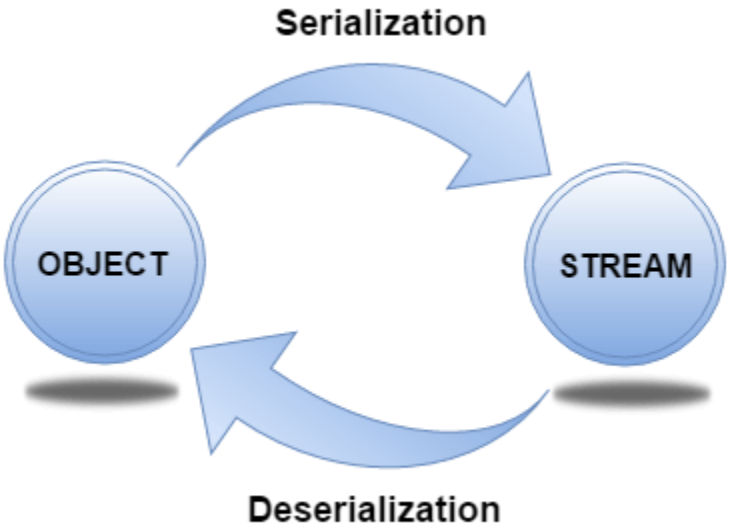
i/o streams and object serialization in java

- Serialization in Java is a mechanism of *writing the state of an object into a byte-stream*. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.
- The reverse operation of serialization is called *deserialization* where byte-stream is converted into an object.
- The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

- For serializing the object, we call the writeObject() method of *ObjectOutputStream* class, and for deserialization we call the readObject() method of *ObjectInputStream* class.

ADVANTAGES OF JAVA SERIALIZATION

It is mainly used to travel object's state on the network (that is known as marshalling).



JAVA.IO.SERIALIZABLE INTERFACE

- Serializable is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The Cloneable and Remote are also marker interfaces.
- The Serializable interface must be implemented by the class whose object needs to be persisted.
- The String class and all the wrapper classes implement the *java.io.Serializable* interface by default.

Student.java

```
import java.io.Serializable;
public class Student implements Serializable{
 int id;
 String name;
 public Student(int id, String name) {
this.id = id;
  this.name = name;
```

```
 }
}
```

In the above example, *Student* class implements Serializable interface. Now its objects can be converted into stream. The main class implementation of is showed in the next code.

OBJECTOUTPUTSTREAM CLASS

The ObjectOutputStream class is used to write primitive data types, and Java objects to an OutputStream. Only objects that support the java.io.Serializable interface can be written to streams.

Constructor

public ObjectOutputStream(OutputStream out) throws IOException
{
}
	It creates an ObjectOutputStream that writes to the specified OutputStream.

Important Methods

Method	Description
1) public final void writeObject(Object obj) throws IOException {}	It writes the specified object to the ObjectOutputStream.
2) public void flush() throws IOException {}	It flushes the current output stream.
3) public void close() throws IOException {}	It closes the current output stream.

OBJECTINPUTSTREAM CLASS

An ObjectInputStream deserializes objects and primitive data written using an ObjectOutputStream.

Constructor

1) public ObjectInputStream(InputStream in) throws IOException {}	It creates an ObjectInputStream that reads from the specified InputStream.

EXAMPLE OF JAVA SERIALIZATION

We are going to serialize the object of *Student* class from above code. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

Persist.java

```
import java.io.*;
class Persist{
 public static void main(String args[]){
  try{
  //Creating the object
  Student s1 =new Student(211,"ravi");
  //Creating stream and writing the object
  FileOutputStream fout=new FileOutputStream("f.txt");
  ObjectOutputStream out=new ObjectOutputStream(fout);
  out.writeObject(s1);
  out.flush();
  //closing the stream
  out.close();
  System.out.println("success");
  }catch(Exception e){System.out.println(e);}
 }
}
```

Example of Java Deserialization

- Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization.
- Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization.

Depersist.java

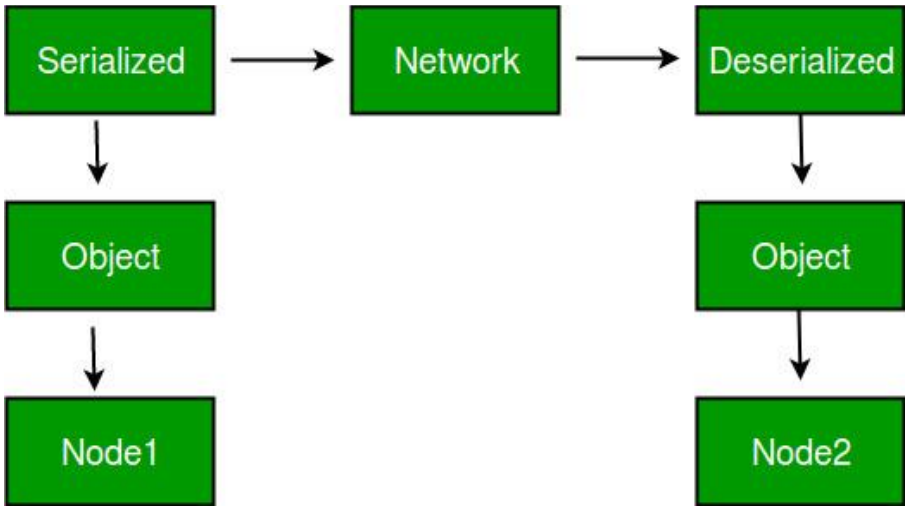
```
 import java.io.*;
class Depersist{
 public static void main(String args[]){
  try{
  //Creating stream to read the object
```

```
        ObjectInputStream in=new ObjectInputStream(new FileInputStream("f.txt"))
       Student s=(Student)in.readObject();
//printing the data of the serialized object
        System.out.println(s.id+" "+s.name);
        //closing the stream
        in.close();
        }catch(Exception e){System.out.println(e);}
       }
       }
```

Advantages of Serialization

1. To save/persist state of an object.

2. To travel an object across a network.



Only the objects of those classes can be serialized which are implementing java.io.Serializable interface.

GENERIC CLASS IN JAVA

- The Java Generics programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.
- Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

ADVANTAGE OF JAVA GENERICS

There are mainly 3 advantages of generics. They are as follows:

1) Type-safety: We can hold only a single type of objects in generics.

Without Generics, we can store any type of objects.

```
List list = new ArrayList();
list.add(10);
list.add("10");
```

With Generics, it is required to specify the type of object we need to store.

```
List<Integer> list = new ArrayList<Integer>();
list.add(10);
list.add("10");// compile-time error
```

2) Type casting is not required: There is no need to typecast the object.

Before Generics, we need to type cast.

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);//typecasting
```

After Generics, we don't need to typecast the object.

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

3) Compile-Time Checking: It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32);//Compile Time Error
```

Syntax to use generic collection

```
ClassOrInterface<Type>
```

Example to use Generics in java

```
ArrayList<String>
```

FULL EXAMPLE OF GENERICS IN JAVA

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

```
import java.util.*;
class TestGenerics1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();
list.add("rahul");
list.add("jai");
//list.add(32);//compile time error

String s=list.get(1);//type casting is not required
System.out.println("element is: "+s);

Iterator<String> itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

```
import java.util.*;
class TestGenerics1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();
list.add("rahul");
list.add("jai");
//list.add(32);//compile time error

String s=list.get(1);//type casting is not required
System.out.println("element is: "+s);

Iterator<String> itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

GENERIC CLASS

- A class that can refer to any type is known as a generic class. Here, we are using the T type parameter to create the generic class of specific type.
- simple example to create and use the generic class.

CREATING A GENERIC CLASS:

```
class MyGen<T>{
T obj;
void add(T obj){this.obj=obj;}
T get(){return obj;}
}
```

The T type indicates that it can refer to any type (like String, Integer, and Employee). The type you specify for the class will be used to store and retrieve the data.

Using generic class:

The code to use the generic class.

```
class TestGenerics3{
public static void main(String args[]){
MyGen<Integer> m=new MyGen<Integer>();
m.add(2);
//m.add("vivek");//Compile time error
System.out.println(m.get());
}}
```

TYPE PARAMETERS

The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:

T - Type

E - Element

K - Key

N - Number

V - Value

GENERIC METHOD

- Generic class, we can create a generic method that can accept any type of arguments. Here, the scope of arguments is limited to the method where it is declared.
- It allows static as well as non-static methods.
- simple example of java generic method to print array elements. We are using here E to denote the element.

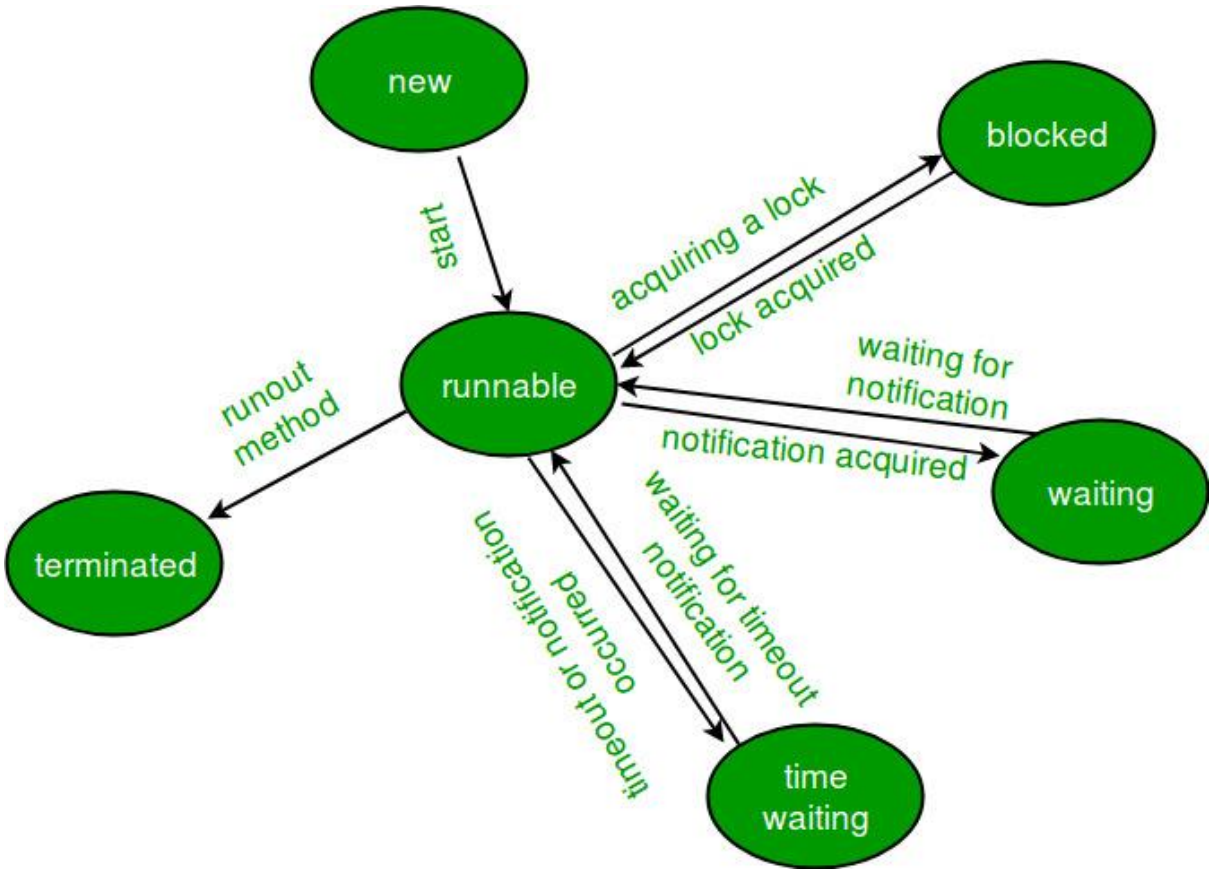
```
public class TestGenerics4{

  public static < E > void printArray(E[] elements) {
      for ( E element : elements){
         System.out.println(element );
       }
       System.out.println();
   }
   public static void main( String args[] ) {
      Integer[] intArray = { 10, 20, 30, 40, 50 };
      Character[] charArray = { 'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };

      System.out.println( "Printing Integer Array" );
      printArray( intArray  );

     System.out.println( "Printing Character Array" );
      printArray( charArray );
   }
}
```

LIFECYCLE AND STATES OF A THREAD IN JAVA

A thread in Java at any point of time exists in any one of the following states. A thread lies only in one of the shown states at any instant:

1. New
2. Runnable
3. Blocked
4. Waiting
5. Timed Waiting
6. Terminated

The diagram shown below represents various states of a thread at any instant in time.



LIFE CYCLE OF A THREAD

1. **New Thread**: When a new thread is created, it is in the new state. The thread has not yet started to run when the thread is in this state. When a thread lies in the new state, its code is yet to be run and hasn't started to execute**.**
2. **Runnable State**: A thread that is ready to run is moved to a runnable state. In this state, a thread might actually be running or it might be ready to run at any instant of time. It is the responsibility of the thread scheduler to give the thread, time to run.

   A multi-threaded program allocates a fixed amount of time to each individual thread. Each and every thread runs for a short while and then pauses and relinquishes the CPU to another thread so that other threads can get a chance to run. When this happens, all such threads that are ready to run, waiting for the CPU and the currently running thread lie in a runnable state.
3. **Blocked/Waiting state**: When a thread is temporarily inactive, then it's in one of the following states:
   - Blocked

- Waiting

4. **Timed Waiting**: A thread lies in a timed waiting state when it calls a method with a time-out parameter. A thread lies in this state until the timeout is completed or until a notification is received. For example, when a thread calls sleep or a conditional wait, it is moved to a timed waiting state.
5. **Terminated State**: A thread terminates because of either of the following reasons:
    - Because it exits normally. This happens when the code of the thread has been entirely executed by the program.
    - Because there occurred some unusual erroneous event, like segmentation fault or an unhandled exception.

Implementing the Thread States in Java

- In Java, to get the current state of the thread, use Thread.getState() method to get the current state of the thread.
- Java provides java.lang.Thread.State class that defines the ENUM constants for the state of a thread, as a summary of which is given below:

1. NEW

Declaration: public static final Thread.State NEW

Description: Thread state for a thread that has not yet started.

2. RUNNABLE

Declaration: public static final Thread.State RUNNABLE

Description: Thread state for a runnable thread. A thread in the runnable state is executing in the Java virtual machine but it may be waiting for other resources from the operating system such as a processor.

3. **BLOCKED**

Declaration: public static final Thread.State BLOCKED

Description: Thread state for a thread blocked waiting for a monitor lock. A thread in the blocked state is waiting for a monitor lock to enter a synchronized block/method or reenter a synchronized block/method after calling Object.wait().

4. WAITING

Declaration: public static final Thread.State WAITING

Description: Thread state for a waiting thread. Thread state for a waiting thread. A thread is in the waiting state due to calling one of the following methods:

- Object.wait with no timeout
- Thread.join with no timeout
- LockSupport.park

5. TIMED WAITING

Declaration: public static final Thread.State TIMED_WAITING

Description: Thread state for a waiting thread with a specified waiting time. A thread is in the timed waiting state due to calling one of the following methods with a specified positive waiting time:

- Thread.sleep
- Object.wait with timeout
- Thread.join with timeout
- LockSupport.parkNanos
- LockSupport.parkUntil

6. TERMINATED

Declaration: public static final Thread.State TERMINATED

Description: Thread state for a terminated thread. The thread has completed execution

```
// Java program to demonstrate thread states
class thread implements Runnable {
    public void run()
    {
        // moving thread2 to timed waiting state
        try {
            Thread.sleep(1500);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println(
            "State of thread1 while it called join() method on thread2 -"
            + Test.thread1.getState());
        try {
            Thread.sleep(200);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```
    }
}

public class Test implements Runnable {
    public static Thread thread1;
    public static Test obj;

    public static void main(String[] args)
    {
        obj = new Test();
        thread1 = new Thread(obj);

        // thread1 created and is currently in the NEW
        // state.
        System.out.println(
            "State of thread1 after creating it - "
            + thread1.getState());
        thread1.start();

        // thread1 moved to Runnable state
        System.out.println(
            "State of thread1 after calling .start() method on it - "
            + thread1.getState());
    }

    public void run()
    {
        thread myThread = new thread();
        Thread thread2 = new Thread(myThread);

        // thread1 created and is currently in the NEW
        // state.
        System.out.println(
            "State of thread2 after creating it - "
            + thread2.getState());
        thread2.start();

        // thread2 moved to Runnable state
        System.out.println(
            "State of thread2 after calling .start() method on it - "
            + thread2.getState());

        // moving thread1 to timed waiting state
        try {
            // moving thread1 to timed waiting state
            Thread.sleep(200);
        }
        catch (InterruptedException e) {
```

```
            e.printStackTrace();
        }
        System.out.println(
            "State of thread2 after calling .sleep() method on it - "
            + thread2.getState());

        try {
            // waiting for thread2 to die
            thread2.join();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(
            "State of thread2 when it has finished it's execution - "
            + thread2.getState());
    }
}
```

Thread synchronization

- Synchronization in Java is the capability to control the access of multiple threads to any shared resource.
- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Here, we will discuss only thread synchronization.

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. **Mutual Exclusive**
    0. Synchronized method.
    1. Synchronized block.
    2. Static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

Concept of Lock in Java

- Synchronization is built around an internal entity known as the lock or monitor.
- Every object has a lock associated with it. By convention, a thread that needs consistent access to an object's fields has to acquire the object's lock before accessing them, and then release the lock when it's done with them.
- From Java 5 the package java.util.concurrent.locks contains several lock implementations.

Understanding the problem without Synchronization

In this example, there is no synchronization, so output is inconsistent. Let's see the example:

TestSynchronization1.java

```
class Table{
void printTable(int n){//method not synchronized
  for(int i=1;i<=5;i++){
    System.out.println(n*i);
    try{
     Thread.sleep(400);
    }catch(Exception e){System.out.println(e);}
  }
```

```
 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Java Synchronized Method

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.

- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

TestSynchronization2.java

```
//example of java synchronized method
class Table{
 synchronized void printTable(int n){//synchronized method
   for(int i=1;i<=5;i++){
     System.out.println(n*i);
     try{
      Thread.sleep(400);
     }catch(Exception e){System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}
```

```
public class TestSynchronization2{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

JAVA NETWORKING

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. Sharing resources
2. Centralize software management

The java.net package supports two protocols,

1. TCP: Transmission Control Protocol provides reliable communication between the sender and receiver. TCP is used along with the Internet Protocol referred as TCP/IP.
2. UDP: User Datagram Protocol provides a connection-less protocol service by allowing packet of data to be transferred along two or more nodes

JAVA NETWORKING TERMINOLOGY

The widely used Java networking terminologies are given below:

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

1)IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1 . It is composed of octets that range from 0 to 255.

It is a logical address that can be changed.

2) Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP etc.

3) Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications.

The port number is associated with the IP address for communication between two applications.

4) MAC Address

MAC (Media Access Control) address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC address.

For example, an ethernet card may have a MAC address of 00:0d:83::b1:c0:8e.

5) Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP.

But, in connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

6) Socket

A socket is an endpoint between two way communications.

Visit next page for Java socket programming.

JAVA.NET PACKAGE

The java.net package can be divided into two sections:

1. A Low-Level API: It deals with the abstractions of addresses i.e. networking identifiers, Sockets i.e. bidirectional data communication mechanism and Interfaces i.e. network interfaces.
2. A High-Level API: It deals with the abstraction of URIs i.e. Universal Resource Identifier, URLs i.e. Universal Resource Locator, and Connections i.e. connections to the resource pointed by URLs.