UNIT – III

Unit – 3 Syllabus :

Exception Handling: Concepts of Exception handling, types of exceptions, usage of try, catch, throw, throws and finally keywords, multiple catch clauses, nested try, Built-in exceptions, creating own exception sub classes.
Multithreading: The Java Thread model, thread life cycle, Thread class, Runnable interface, creating multiple threads, Synchronization, Inter Thread Communication, Deadlock.
Applets: Concepts of Applets, life cycle of an applet, creating applets
Event Handling: Events, Event sources, Event classes, Event Listeners, Delegation event model, handling events.

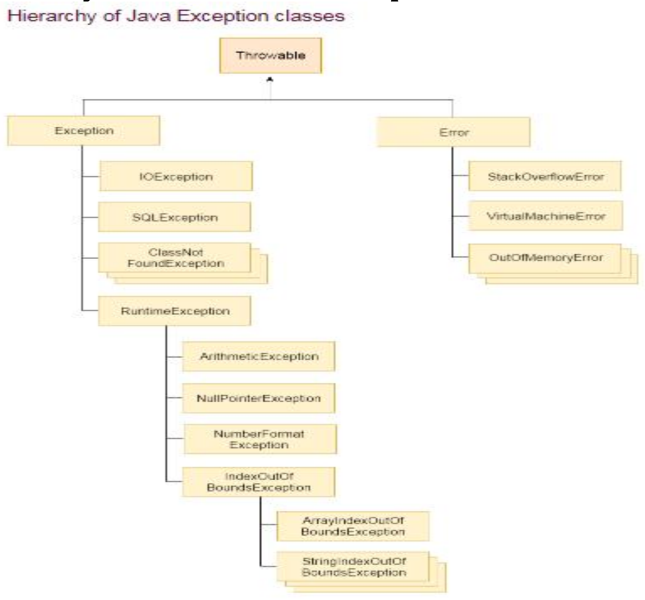
Exception Handling

Concepts of Exception handling:

Exception : Exception is an abnormal condition.
Exception Handling : Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.
Advantages of Exception Handling :

- The core advantage of exception handling is to maintain the normal flow of the application.
- Exception normally disrupts the normal flow of the application that is why we use exception handling.

The java.lang.Throwable class is the root class of Java Exception hierarchy which is inherited by two subclasses: Exception and Error.



Types of Exceptions :

There are mainly two types of exceptions: checked and unchecked
Checked Exception
Checked exceptions are checked at compile-time.
E.g: IOException, SQLException etc.
Unchecked Exception
Unchecked exceptions are not checked at compile-time rather they are checked at runtime.
E.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.
1) Scenario where ArithmeticException occurs
If we divide any number by zero, there occurs an ArithmeticException.
```
int a=50/0;//ArithmeticException
```
2) Scenario where NullPointerException occurs
If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.
```
String s=null;
System.out.println(s.length());//NullPointerException
```
3) Scenario where NumberFormatException occurs
The wrong formatting of any value, may occur NumberFormatException.
Suppose if we have a string variable that have characters, converting this variable into digit will occur NumberFormatException.
```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```
4) Scenario where ArrayIndexOutOfBoundsException occurs
If we are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:
```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

Java Exception Handling Keywords :
There are 5 keywords used in java exception handling.

- Try
- Catch
- Finally
- Throw
- Throws

General form of an Exception Handling Block :
```
try {
// block of code to monitor for errors
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
```

```
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// ...
finally {
// block of code to be executed after try block ends
}
```

Java try-catch :

Java try block is used to enclose the code that might throw an exception. It must be used within the method.
Java try block must be followed by either catch or finally block.
Syntax of try-catch block

```
try{
//code that may throw exception
}
catch(Exception_class_Name ref)
{
}
```

Java catch block is used to handle the Exception. It must be used after the try block only.
We can use multiple catch block with a single try.
Example :

```
public class Testtrycatch1
{
 public static void main(String args[])
{
  try
{
     int data=50/0;
  }
catch(ArithmeticException e)
{
System.out.println(e);
}
  System.out.println("rest of the code...");
}
}
```

Output :

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Java Multi catch block :

```
public class TestMultipleCatchBlock{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[4]=30/0;
  }
  catch(ArithmeticException e){System.out.println("task1 is completed");}
  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
  catch(Exception e){System.out.println("common task completed");}

  System.out.println("rest of the code...");
 }
}
```

Output :

```
task1 completed
rest of the code…
```

Note:

- At a time only one Exception is occured and at a time only one catch block is executed.
- All catch blocks must be ordered from most specific to most general i.e. catch for ArithmeticException must come before catch for Exception .
- For each try block there can be zero or more catch blocks, but only one finally block.

Java Nested try block :

The try block within a try block is known as nested try block in java.

Why use nested try block ?

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

Syntax :

```
....
try
{
   statement 1;
   statement 2;
   try
   {
      statement 1;
      statement 2;
   }
   catch(Exception e)
   {
   }
}
```

```
catch(Exception e)
{
}
....
```

Example :

```
class Excep6{
public static void main(String args[]){
try{
try{
System.out.println("going to divide");
int b=39/0;
}catch(ArithmeticException e){System.out.println(e);}

try{
int a[]=new int[5];
a[5]=4;
}catch(ArrayIndexOutOfBoundsException e){System.out.println(e);}

System.out.println("other statement");
}catch(Exception e){System.out.println("handeled");}

System.out.println("normal flow..");
}
}
```

Output :

```
going to divide
java.lang.ArithmeticException: / by zero
java.lang.ArrayIndexOutOfBoundsException: 5
other statement
normal flow..
```

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.

Usage of Java finally :

```
class TestFinallyBlock{
  public static void main(String args[]){
  try{
   int data=25/5;
   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...");
 }
}
```

Output :

```
5
finally block is always executed rest of the code...
```

Java throw Keyword :

- The Java throw keyword is used to explicitly throw an exception.
- We can throw either checked or uncheked exception in java by throw keyword.
- The throw keyword is mainly used to throw custom exception.
- The syntax of java throw keyword is given below.

```
throw exception;
throw new IOException("sorry device error”);
```

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);
       }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:not valid
```

Java throws Keyword :

- The Java throws keyword is used to declare an exception.
- It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
- Syntax of java throws

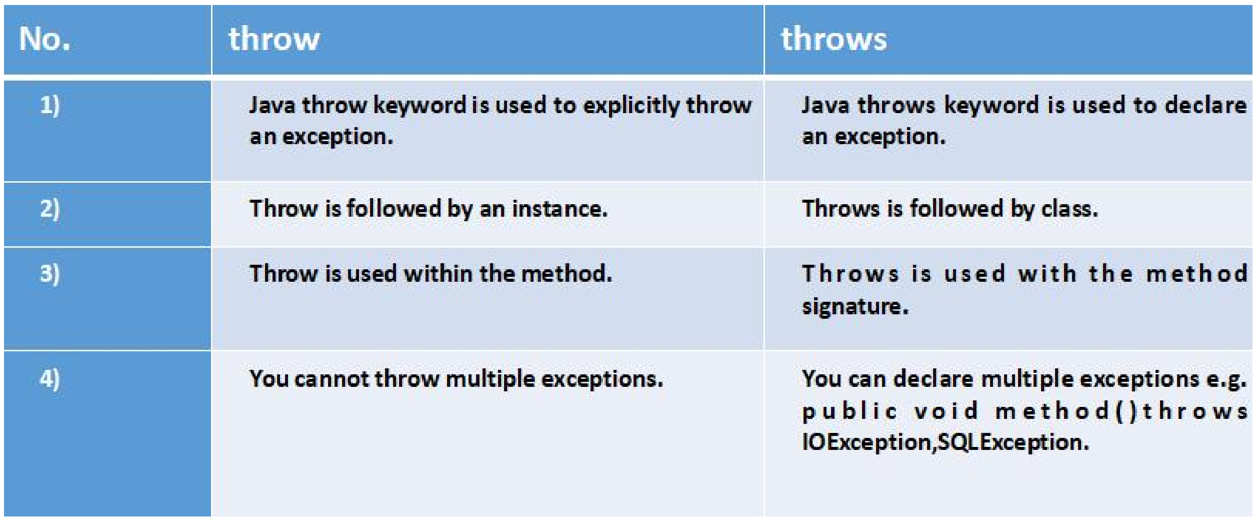
```
return_type method_name() throws exception_class_name{
//method code
}
```

Example :

```
import java.io.IOException;
class Testthrows1{
 void m()throws IOException{
  throw new IOException("device error");//checked exception
 }
 void n()throws IOException{
  m();
 }
 void p(){
  try{
   n();
  }catch(Exception e){System.out.println("exception handled");}
 }
 public static void main(String args[]){
  Testthrows1 obj=new Testthrows1();
  obj.p();
  System.out.println("normal flow...");
 }
}
```

Output :

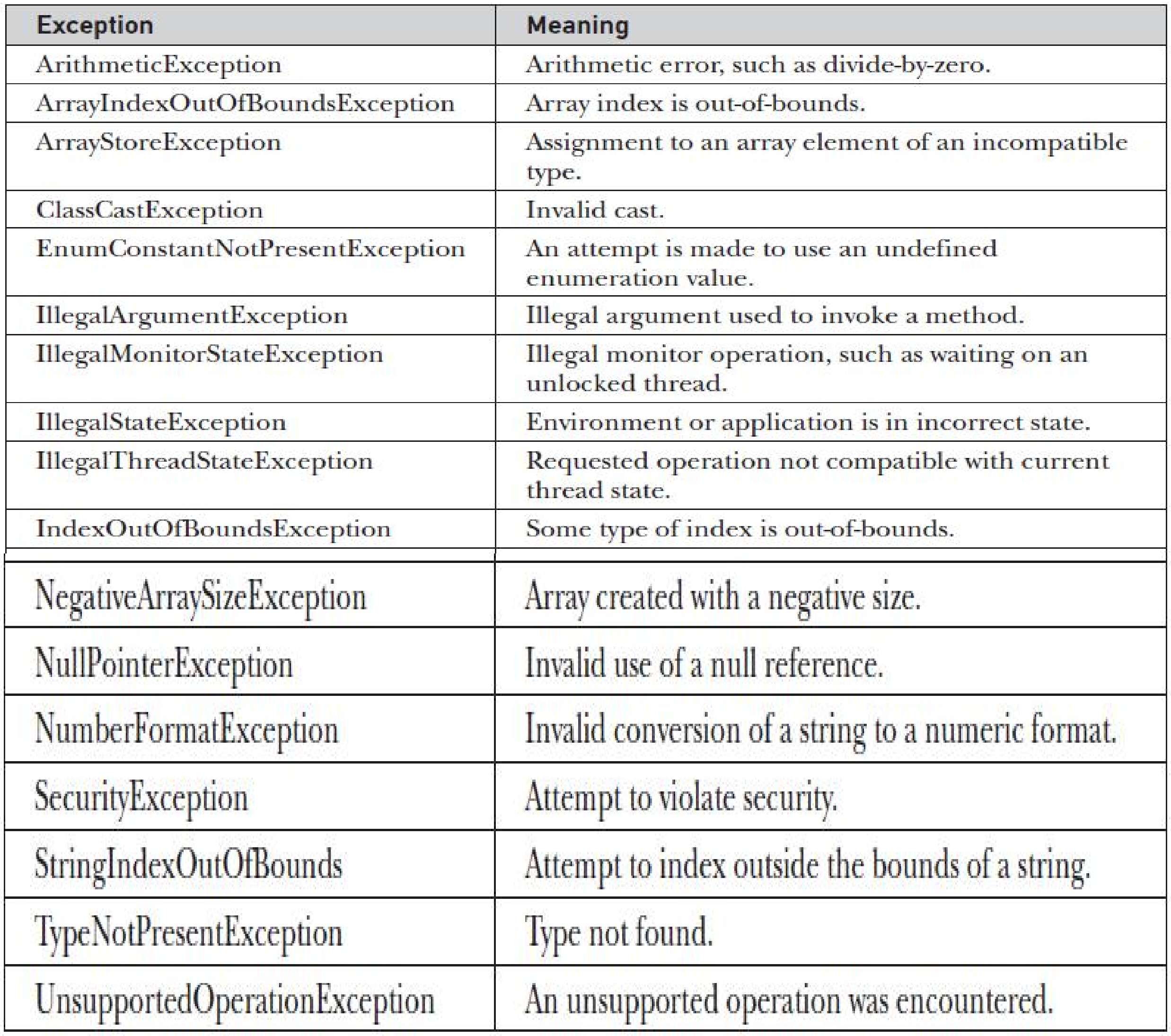
```
exception handled
normal flow...
```



No.	throw	throws
1)	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2)	Throw is followed by an instance.	Throws is followed by class.
3)	Throw is used within the method.	Throws is used with the method signature.
4)	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

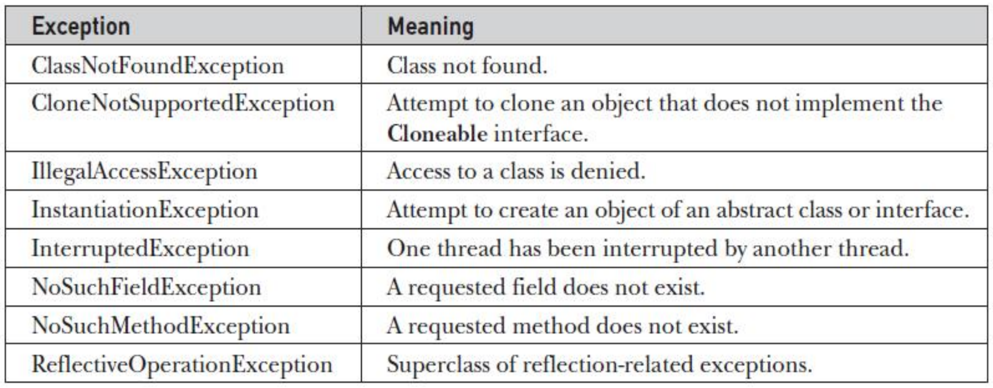
Java’s Built-in Exceptions

Java’s Unchecked RuntimeException Subclasses Defined in java.lang



Exception	Meaning
ArithmeticException	Arithmetic error, such as divide-by-zero.
ArrayIndexOutOfBoundsException	Array index is out-of-bounds.
ArrayStoreException	Assignment to an array element of an incompatible type.
ClassCastException	Invalid cast.
EnumConstantNotPresentException	An attempt is made to use an undefined enumeration value.
IllegalArgumentException	Illegal argument used to invoke a method.
IllegalMonitorStateException	Illegal monitor operation, such as waiting on an unlocked thread.
IllegalStateException	Environment or application is in incorrect state.
IllegalThreadStateException	Requested operation not compatible with current thread state.
IndexOutOfBoundsException	Some type of index is out-of-bounds.
NegativeArraySizeException	Array created with a negative size.
NullPointerException	Invalid use of a null reference.
NumberFormatException	Invalid conversion of a string to a numeric format.
SecurityException	Attempt to violate security.
StringIndexOutOfBounds	Attempt to index outside the bounds of a string.
TypeNotPresentException	Type not found.
UnsupportedOperationException	An unsupported operation was encountered.

Java’s Checked Exceptions Defined in java.lang



Exception	Meaning
ClassNotFoundException	Class not found.
CloneNotSupportedException	Attempt to clone an object that does not implement the **Cloneable** interface.
IllegalAccessException	Access to a class is denied.
InstantiationException	Attempt to create an object of an abstract class or interface.
InterruptedException	One thread has been interrupted by another thread.
NoSuchFieldException	A requested field does not exist.
NoSuchMethodException	A requested method does not exist.
ReflectiveOperationException	Superclass of reflection-related exceptions.

Java Custom Exception (User Defined Exception)

- Although Java's built-in exceptions handle most common errors, we will probably want to create our own exception types to handle situations specific to our applications.
- This is quite easy to do: just define a subclass of Exception.
- If we are creating our 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.

Example :

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
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 occured: "+m);}
      System.out.println("rest of the code...");
  }
}
```

Output :

```
Exception occured: InvalidAgeException:not valid
rest of the code...
```

Example II :

```
class MyException extends Exception{
   String str1;
   MyException(String str2) {
          str1=str2;
   }
   public String toString(){
          return ("MyException Occurred: "+str1) ;
   }
}
class Example1{
   public static void main(String args[]){
          try{
                  System.out.println("Starting of try block");
                  throw new MyException("This is My error Message");
          }
          catch(MyException exp){
                  System.out.println("Catch Block") ;
                  System.out.println(exp) ;
          }
   }
}
```

Example III :

1.
   ```
   class InvalidProductException extends Exception
   {
      public InvalidProductException(String s)
      {
             super(s);
      }
   }
    public class Example1{
     void productCheck(int weight) throws InvalidProductException{
           if(weight<100){
                   throw new InvalidProductException("Product Invalid");
           }
     }

      public static void main(String args[]){
           Example1 obj = new Example1();
         try
         {
            obj.productCheck(60);
         }
         catch (InvalidProductException ex)
         {
            System.out.println("Caught the exception");
            System.out.println(ex.getMessage());
         }
      }
   }
   ```

   NOTE:

1. User-defined exception must extend Exception class.
2. The exception is thrown using throw keyword.
3. If we are throwing an exception from a method, we should use throws clause in the method signature otherwise we will get compilation error saying that “unhandled exception in method”.

Multithreading in Java

Multithreading in java is a process of executing multiple threads simultaneously. Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking. But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process. Java Multithreading is mostly used in games, animation etc.

Advantages of java multithreading:

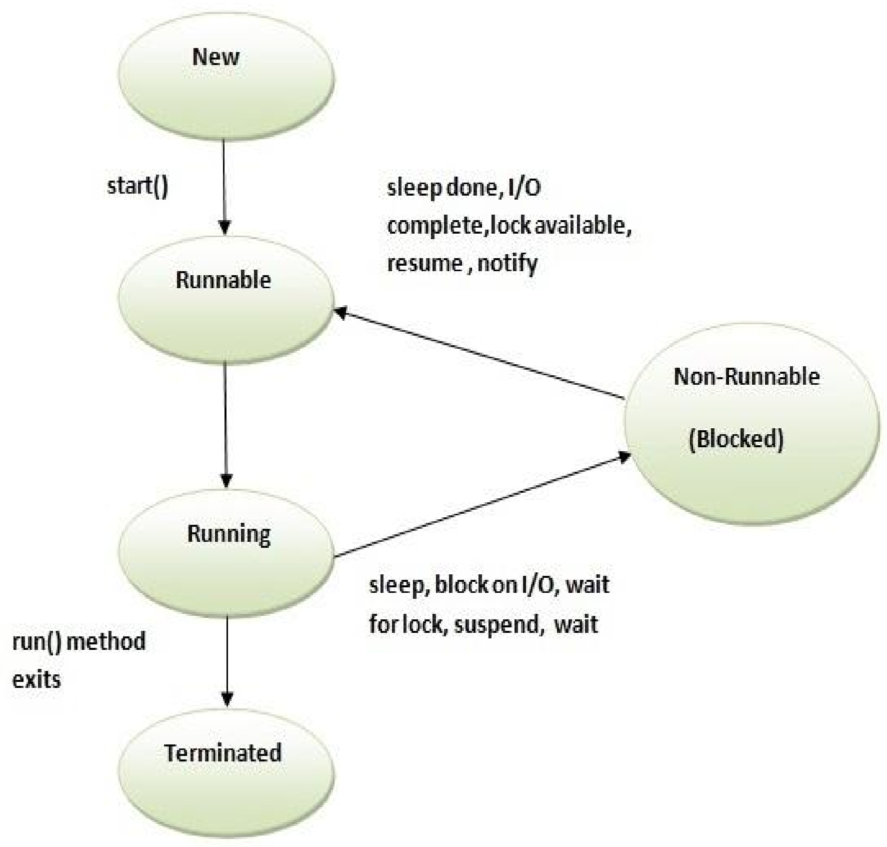
1) It doesn't block the user because threads are independent and we can perform multiple operations at same time.
2) We can perform many operations together so it saves time.
3) Threads are independent so it doesn't affect other threads if exception occur in a single thread.

The Java Thread Model:

- A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.
- Threads are independent, if there occurs exception in one thread, it doesn't affect other threads.
- Thread shares a common memory area.
- **Note**: At a time one thread is executed only.

The Java Thread Life Cycle:

- A thread can be in one of the five states.
- The life cycle of the thread in java is controlled by JVM.
- The java thread states are as follows:
    1. New
    2. Runnable
    3. Running
    4. Non-Runnable (Blocked)
    5. Terminated

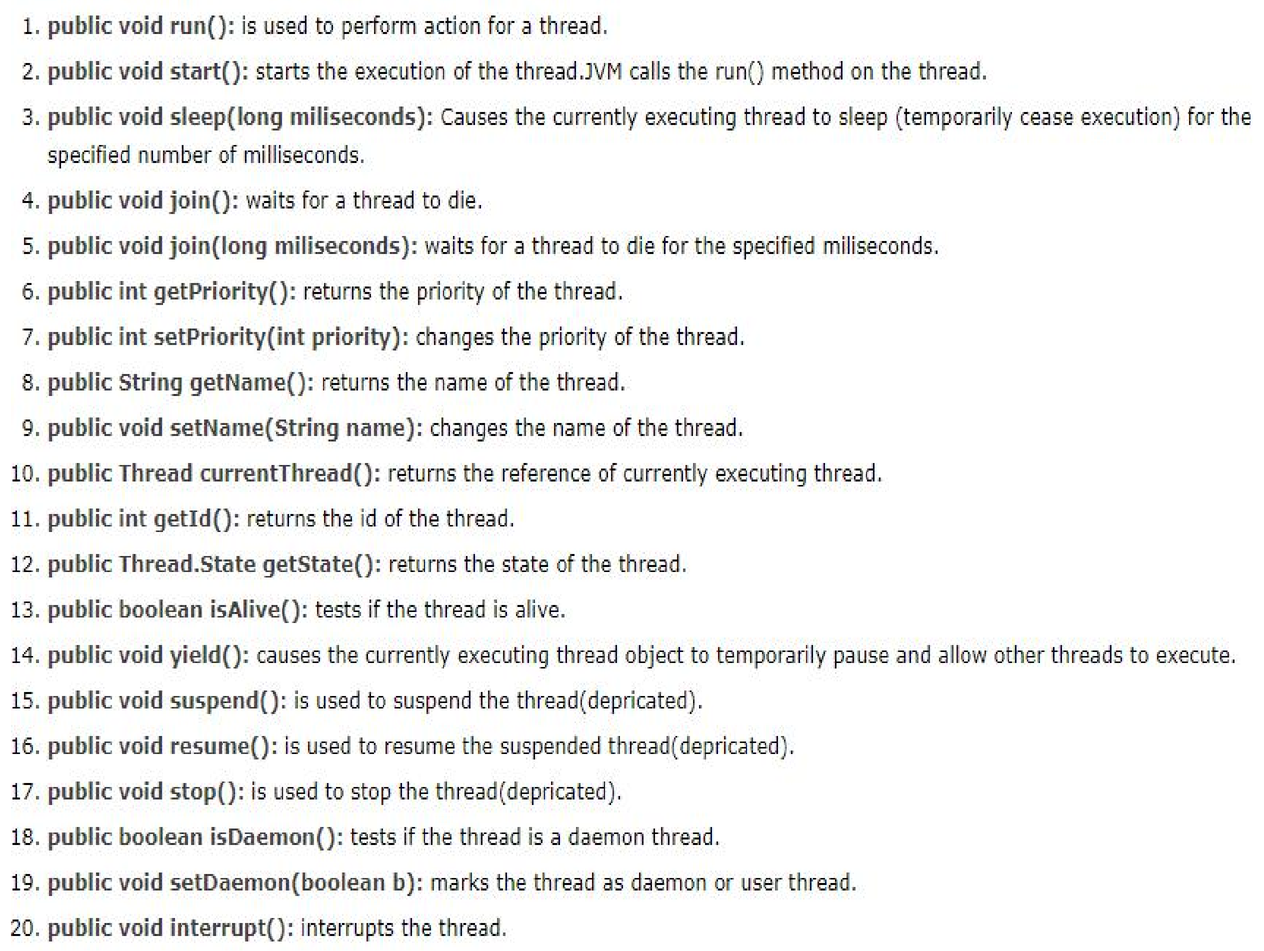

New
start()
Runnable
sleep done, I/O complete,lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

1) New
The thread is in new state if we create an instance of Thread class but before the invocation of start() method.
2) Runnable
The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
3) Running
The thread is in running state if the thread scheduler has selected it.
4) Non-Runnable (Blocked)
This is the state when the thread is still alive, but is currently not eligible to run.
5) Terminated
A thread is in terminated or dead state when its run() method exits.

Creation of Threads:

There are two ways to create a thread:
1) By extending Thread class
2) By implementing Runnable interface.

Before we get to know how to create a thread, let us look into some of the methods in Threads:



1. **public void run():** is used to perform action for a thread.
2. **public void start():** starts the execution of the thread.JVM calls the run() method on the thread.
3. **public void sleep(long miliseconds):** Causes the currently executing thread to sleep (temporarily cease execution) for the specified number of milliseconds.
4. **public void join():** waits for a thread to die.
5. **public void join(long miliseconds):** waits for a thread to die for the specified miliseconds.
6. **public int getPriority():** returns the priority of the thread.
7. **public int setPriority(int priority):** changes the priority of the thread.
8. **public String getName():** returns the name of the thread.
9. **public void setName(String name):** changes the name of the thread.
10. **public Thread currentThread():** returns the reference of currently executing thread.
11. **public int getId():** returns the id of the thread.
12. **public Thread.State getState():** returns the state of the thread.
13. **public boolean isAlive():** tests if the thread is alive.
14. **public void yield():** causes the currently executing thread object to temporarily pause and allow other threads to execute.
15. **public void suspend():** is used to suspend the thread(depricated).
16. **public void resume():** is used to resume the suspended thread(depricated).
17. **public void stop():** is used to stop the thread(depricated).
18. **public boolean isDaemon():** tests if the thread is a daemon thread.
19. **public void setDaemon(boolean b):** marks the thread as daemon or user thread.
20. **public void interrupt():** interrupts the thread.

Java Thread Example by extending Thread class

We can extend java. lang.Thread class to create our own java thread class and override run() method. Then we can create it's object and call start() method to execute our custom java thread class run method.

Example:

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
}
```

Output: thread is running...

Java Thread Example by implementing Runnable interface

- To make a class runnable, we can implement java.lang.Runnable interface and provide implementation in public void run() method.
- To use this class as Thread, we need to create a Thread object by passing object of this runnable class and then call start() method to execute the run() method in a separate thread.

Example:

```
class Multi3 implements Runnable
{
public void run()
{
System.out.println("thread is running...");
}
public static void main(String args[])
{
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);
t1.start();
 }
}
```
Output: thread is running...

Some of the methods in Java Threads:

1. The Sleep() method:
The sleep() method of Thread class is used to sleep a thread for the specified amount of time.
Syntax:
public static void sleep(long miliseconds)throws InterruptedException
Example:
```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    try{
Thread.sleep(500);
}
catch(InterruptedException e)
{System.out.println(e);}
    System.out.println(i);
  }
 }
 public static void main(String args[]){
  TestSleepMethod1 t1=new TestSleepMethod1();
  TestSleepMethod1 t2=new TestSleepMethod1();
  t1.start();
  t2.start();
 }
}
```

2. The Join() method:
Join method in Java allows one thread to wait until another thread completes its execution. In simpler words, it means it waits for the other thread to die. It has a void type and throws InterruptedException. It is mostly used in Inter-Thread Communication.
Syntax:

```
public void join()throws InterruptedException

public void join(long milliseconds)throws InterruptedException
```

Example:

```
class TestJoinMethod1 extends Thread{
 public void run(){
  for(int i=1;i<=5;i++){
   try{
    Thread.sleep(500);
   }catch(Exception e){System.out.println(e);}
  System.out.println(i);
  }
 }
public static void main(String args[]){
 TestJoinMethod1 t1=new TestJoinMethod1();
 TestJoinMethod1 t2=new TestJoinMethod1();
 TestJoinMethod1 t3=new TestJoinMethod1();
 t1.start();
 try{
  t1.join();
 }catch(Exception e){System.out.println(e);}
 t2.start();
 t3.start();
 }
}
```

3. getName(),setName(String) and getId() method:
Example:

```
class TestJoinMethod3 extends Thread{
  public void run(){
   System.out.println("running...");
  }
 public static void main(String args[]){
  TestJoinMethod3 t1=new TestJoinMethod3();
  TestJoinMethod3 t2=new TestJoinMethod3();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());
  System.out.println("id of t1:"+t1.getId());
  t1.start();
  t2.start();
  t1.setName("RVR");
  System.out.println("After changing name of t1:"+t1.getName());
 }
}
```

4. The currentThread() method:
The currentThread() method returns a reference to the currently executing thread object.
Example:

```
class TestJoinMethod4 extends Thread{
 public void run(){
  System.out.println(Thread.currentThread().getName());
 }
 }
 public static void main(String args[]){
  TestJoinMethod4 t1=new TestJoinMethod4();
  TestJoinMethod4 t2=new TestJoinMethod4();
  t1.start();
  t2.start();
 }
}
```

Thread Scheduler:

- Thread scheduler in java is the part of the JVM that decides which thread should run.
- There is no guarantee that which runnable thread will be chosen to run by the thread scheduler.
- Only one thread at a time can run in a single process.
- The thread scheduler mainly uses preemptive or time slicing scheduling to schedule the threads.

Java Thread Pool:
Java Thread pool represents a group of worker threads that are waiting for the job and reuse many times.
Advantage of Java Thread Pool

- **Better performance:** It saves time because there is no need to create new thread.
- **Real time usage:** It is used in Servlet and JSP where container creates a thread pool to process the request.

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.
- The synchronization is mainly used to
    - To prevent thread interference.
    - To prevent consistency problem.

Types of Synchronization:

There are two types of synchronization

- Process Synchronization
- Thread Synchronization

Thread synchronization

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

- Mutual Exclusive
    - Synchronized method.
    - Synchronized block.
    - static synchronization.
- Cooperation (Inter-thread communication in java)

Mutual Exclusive:

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

- by synchronized method
- by synchronized block
- by static synchronization

Java Synchronized method:

- If we 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.
- It's example will be discussed in Inter-Thread Communication.

Inter Thread Communication

- **Inter-thread communication** or **Co-operation** is all about allowing synchronized threads to communicate with each other.
- Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed. It is implemented by following methods of **Object class**:
- wait()
- notify()
- notifyAll()

wait() method:

- Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
- The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Method	Description
public final void wait()throws InterruptedException	waits until object is notified.
public final void wait(long timeout)throws InterruptedException	waits for the specified amount of time.

notify() method:

- Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:
- public final void notify()

notifyAll() method:
Wakes up all threads that are waiting on this object's monitor.
Syntax:
public final void notifyAll()

*****Example of Inter-Thread Communication:

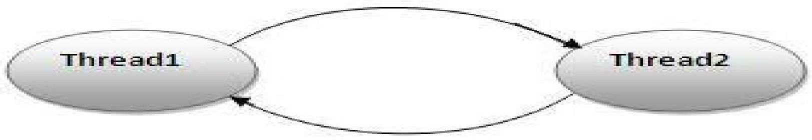
```
import java.util.LinkedList;
public class Main{
  public static void main(String[] args) throws InterruptedException{
    final PC pc = new PC();
    Thread t1 = new Thread(new Runnable(){
      public void run(){
        try {
          pc.produce();
        }
        catch (InterruptedException e) {
          e.printStackTrace();
        }
      }
    });
    Thread t2 = new Thread(new Runnable() {
      public void run(){
            try {
               pc.consume();
            }
            catch (InterruptedException e) {
               e.printStackTrace();
            }
         }
      });
      t1.start();
             t2.start();
      t1.join();
      t2.join();
   }
}
class PC{
      LinkedList<Integer> list = new LinkedList<>();
      int capacity = 2;
      public void produce() throws InterruptedException{
         int value = 0;
         while (true) {
            synchronized (this){
               while (list.size() == capacity)
                  wait();
               System.out.println("Producer produced-"
                              + value);
               list.add(value++);
               notify();
               Thread.sleep(1000);
            }
         }
      }
      public void consume() throws InterruptedException{
         while (true) {
            synchronized (this){
               while (list.size() == 0)
                  wait();
               int val = list.removeFirst();
               System.out.println("Consumer consumed-"
                              + val);
               notify();
```

```
                    Thread.sleep(1000);
                }
            }
        }
}
```
Output:

```
Producer produced-0
Producer produced-1
Consumer consumed-0
Consumer consumed-1
Producer produced-2
Producer produced-3
Consumer consumed-2
Consumer consumed-3
Producer produced-4
Producer produced-5
Consumer consumed-4
Consumer consumed-5
.
.
```

Deadlock:

- Deadlock in java is a part of multithreading.
- Deadlock can occur in a situation when a thread is waiting for an object lock, that is acquired by another thread and second thread is waiting for an object lock that is acquired by first thread.
- Since, both threads are waiting for each other to release the lock, the condition is called deadlock.



Example:

```
public class TestDeadlockExample1{
  public static void main(String[] args){
    final String resource1 = "ratan jaiswal";
    final String resource2 = "vimal jaiswal";
    // t1 tries to lock resource1 then resource2
    Thread t1 = new Thread(){
      public void run(){
          synchronized(resource1){
```

```
       System.out.println("Thread 1: locked resource 1");
       try { Thread.sleep(100);} catch (Exception e) {}
       synchronized (resource2{
        System.out.println("Thread 1: locked resource 2");
       }
      }
     }
    }
    // t2 tries to lock resource2 then resource1
    Thread t2 = new Thread(){
     public void run(){
      synchronized (resource2){
       System.out.println("Thread 2: locked resource 2");
       try { Thread.sleep(100);} catch (Exception e) {}
       synchronized (resource1){
        System.out.println("Thread 2: locked resource 1");
       }
      }
     }
    }
    t1.start();
    t2.start();
  }
}
```
Output:

```
Thread 1: locked resource 1

Thread 2: locked resource 2
```

Applets

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet
- All applets are sub-classes (either directly or indirectly) of *java.applet.Applet* class.

- Applets are not stand-alone programs. Instead, they run within either a web browser or an applet viewer. JDK provides a standard applet viewer tool called applet viewer.
- In general, execution of an applet does not begin at main() method.
- Output of an applet window is not performed by *System.out.println()*. Rather it is handled with various AWT methods, such as *drawString()*.

There are certain differences between Applet and Java Standalone Application that are described below:

- An applet is a Java class that extends the java.applet.Applet class.
- A main() method is not invoked on an applet, and an applet class will not define main().
- Applets are designed to be embedded within an HTML page.
- When a user views an HTML page that contains an applet, the code for the applet is downloaded to the user's machine.
- A JVM is required to view an applet. The JVM can be either a plug-in of the Web browser or a separate runtime environment.
- The JVM on the user's machine creates an instance of the applet class and invokes various methods during the applet's lifetime.
- Applets have strict security rules that are enforced by the Web browser. The security of an applet is often referred to as sandbox security, comparing the applet to a child playing in a sandbox with various rules that must be followed.

Life cycle of an Applet:

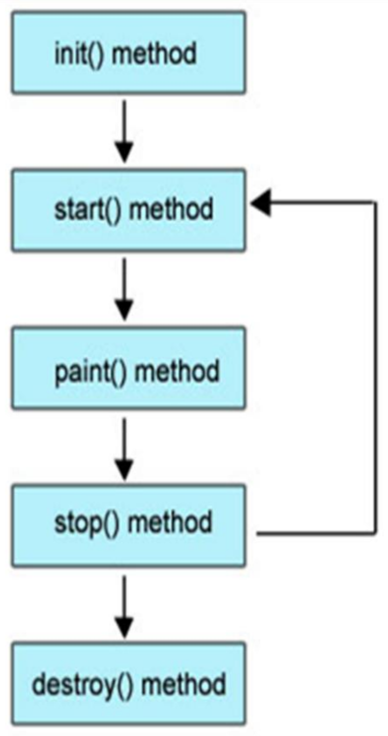


Figure: Life cycle of Applet

Lifecycle methods of applets:

- The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides one life cycle methods for an applet.

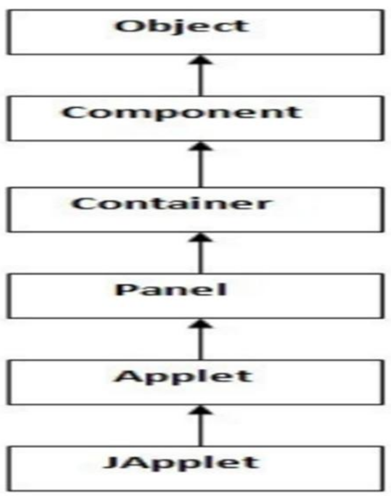
java.applet.Applet class

- For creating any applet java.applet.Applet class must be inherited.
- It provides 4 life cycle methods of applet.
    - **public void init():** is used to initialized the Applet. It is invoked only once.
    - **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
    - **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
    - **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

- The Component class provides 1 life cycle method of applet.
    - **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

Hierarchy of Applet:



How to run an Applet?

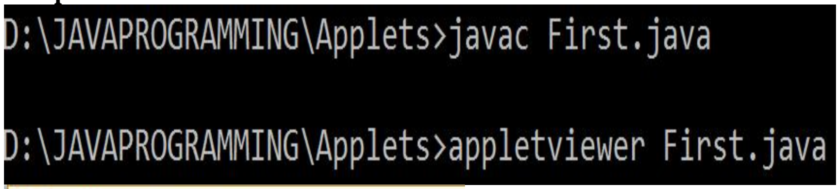
- There are two ways to run an applet
    - By html file.
    - By appletViewer tool

Example:

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
```

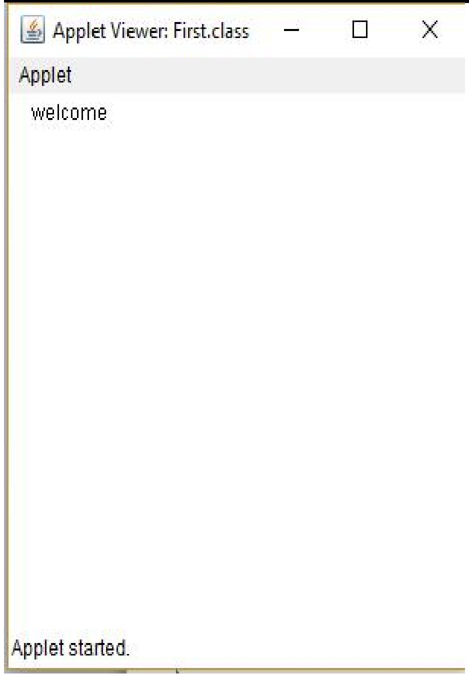
```
g.drawString("welcome",150,150);
}
}
/*<applet code="First.class" width="300" height="300">
</applet>
*/
```

Output:



```
D:\JAVAPROGRAMMING\Applets>javac First.java

D:\JAVAPROGRAMMING\Applets>appletviewer First.java
```



```
welcome
```

```
Applet started.
```

Displaying Graphics in Applet:

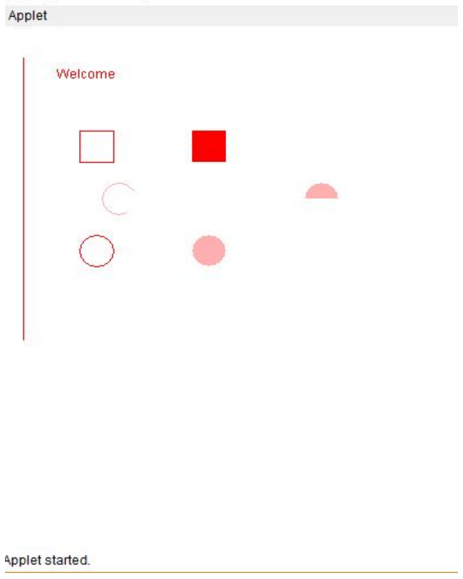
- java.awt.Graphics class provides many methods for graphics programming.
- Commonly used methods of Graphics class:
- **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
- **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
- **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
- **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
- **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
- **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).

- **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
- **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
- **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):**is used to fill a circular or elliptical arc.
- **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
- **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
/*<applet code="GraphicsDemo.class" width="300" height="300"></applet>*/
public class GraphicsDemo extends Applet
{
public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50,50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
}}
```

Output:



Parameter passing in Applet:

- We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

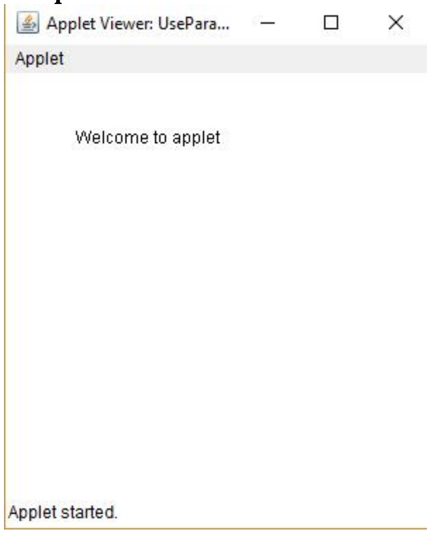
Syntax:

- **public** String getParameter(String parameterName)

Example:

```
import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet{
public void paint(Graphics g){
String str=getParameter("msg");
g.drawString(str,50, 50);
}
}
/*<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>*/
```

Output:



```
Welcome to applet
```

Event Handling:

Event Handling PPT

***