Lecture 2 - Advanced Java Features

Guiliang Liu

The Chinese University of Hong Kong, Shenzhen

CSC-1004: Computational Laboratory Using Java
Course Page: [Click]

Outline

- Java Inheritance
- Java Multi-Threading

Java Inheritance

High-Level Ideas:

- Inheritance in Java is a mechanism in which one object acquires all the properties and behaviors of a parent object.
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.
- **Reusability.** When you inherit from an existing class, you can reuse methods and fields of the parent class.
- **Addition.** you can add new methods and fields in your current class also.

Java Inheritance

Terms used in Inheritance:

- **Class**: A class is a group of objects which have common properties. It is a template or blueprint from which objects are created.
- **Sub Class/Child Class**: Subclass is a class that inherits the other class. It is also called a derived class, extended class, or child class.
- **Super Class/Parent Class**: Superclass is the class from where a subclass inherits the features. It is also called a base class or a parent class.
- **Reusability**: As the name specifies, reusability is a mechanism that facilitates you to reuse the fields and methods of the existing class when you create a new class.

Java Inheritance

Relationship. Inheritance represents the IS-A relationship which is also known as a parent-child relationship.

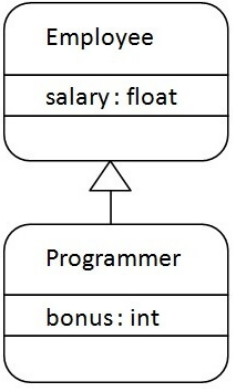
- *The syntax of Java Inheritance.* The extends keyword indicates that you are making a new class that derives from an existing class. The meaning of "extends" is to increase functionality.

```
class Subclass-name extends Superclass-name
{
  //methods and fields
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java Inheritance Example

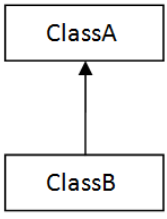
As displayed in the figure below, Programmer is the subclass and Employee is the superclass. The relationship between the two classes is Programmer IS-A Employee. It means that Programmer is a type of Employee.



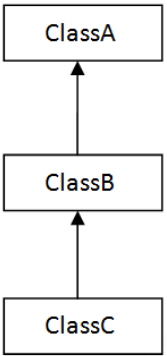

```
class Employee{
 float salary=40000;
}
class Programmer extends Employee{
 int bonus=10000;
 public static void main(String args[]){
   Programmer p=new Programmer();
   System.out.println("Programmer salary is:"+p.salary);
   System.out.println("Bonus of Programmer is:"+p.bonus);
}
}
```

Types of inheritance in java

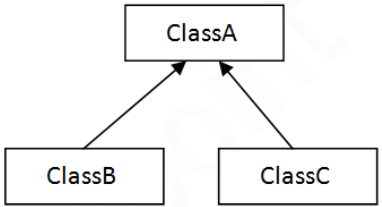
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.



1) Single



2) Multilevel



3) Hierarchical

Single Inheritance Example

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Single Inheritance Example

When a class inherits another class, it is known as a single inheritance. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class TestInheritance{
public static void main(String args[]){
Dog d=new Dog();
d.bark();
d.eat();
}}
```

Output:

barking...

eating...

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance, e.g., BabyDog class inherits the Dog class which again inherits the Animal class.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as multilevel inheritance, e.g., BabyDog class inherits the Dog class which again inherits the Animal class.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class BabyDog extends Dog{
void weep(){System.out.println("weeping...");}
}
class TestInheritance2{
public static void main(String args[]){
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

Output:

weeping...

barking...

eating...

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Hierarchical Inheritance Example

When two or more classes inherit a single class, it is known as hierarchical inheritance, e.g., Dog and Cat classes inherit the Animal class, so there is hierarchical inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Hierarchical Inheritance Example

When two or more classes inherit a single class, it is known as hierarchical inheritance, e.g., Dog and Cat classes inherit the Animal class, so there is hierarchical inheritance.

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void bark(){System.out.println("barking...");}
}
class Cat extends Animal{
void meow(){System.out.println("meowing...");}
}
class TestInheritance3{
public static void main(String args[]){
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}}
```

Output:

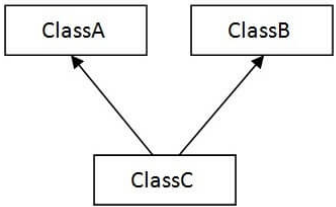
meowing...

eating...

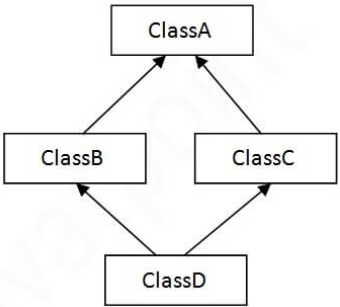
香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Types of inheritance not supported in java

When one class inherits multiple classes, it is known as multiple inheritances. Multiple inheritances are **not supported** in Java.



4) Multiple



5) Hybrid

Types of inheritance not supported in java

Consider a scenario (in 4) above) where A, B, and C are three classes. The C class inherits the A and B classes. If A and B classes have the same method and you call it from the child class object, there will be ambiguity in calling the method of A or B class.

Multithreading in Java

- Multithreading in Java is a process of executing multiple threads simultaneously.
- A thread is a lightweight sub-process, the smallest unit of processing. **Multiprocessing and multithreading**, both are used to achieve multitasking.
- However, we use multithreading than multiprocessing because threads use a **shared memory** area. They don't allocate separate memory areas so save memory, and context-switching between the threads takes less time than the process.

Advantages of Java Multithreading

- It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
- You can perform many operations together, so it saves time.
- Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

1. Process-based Multitasking (Multiprocessing)
    - Each process has an address in memory. In other words, each process allocates a separate memory area.
    - A process is heavyweight.
    - Cost of communication between the process is high.
    - Switching from one process to another requires some time for saving and loading registers, memory maps, updating lists, etc.

Multitasking

Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved in two ways:

1. Process-based Multitasking (Multiprocessing)
2. Thread-based Multitasking (Multithreading)
   - Threads share the same address space.
   - A thread is lightweight.
   - Cost of communication between the thread is low.

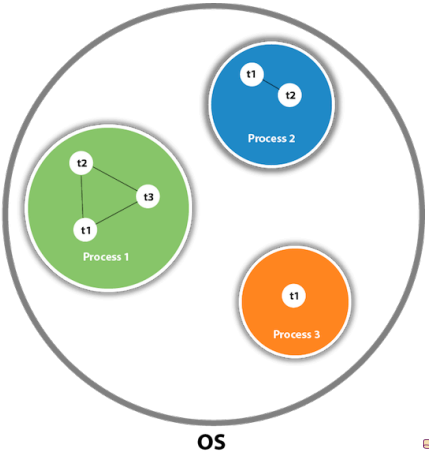
Multitasking

What is Thread in java?

- A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.
- Threads are independent. If there an exception occurs in one thread, it doesn't affect other threads. It uses a shared memory area.

Multitasking

What is Thread in java?



香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:** Whenever a new thread is created, it is always in the new state. For a thread in the new state, the code has not been run yet and thus has not begun its execution.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:** When a thread invokes the start() method, it moves from the new state to the active state. The active state contains two states within it: one is runnable, and the other is running.
   - 2.1 *Runnable*: A thread, that is ready to run is then moved to the runnable state. In the runnable state, the thread may be running or may be ready to run at any given instant of time. The thread scheduler provides the thread time to run.
   - 2.2 *Running*: When the thread gets the CPU, it moves from the runnable to the running state. Generally, the most common change in the state of a thread is from runnable to running and again back to runnable.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:**
3. **Blocked / Waiting:** Whenever a thread is inactive for a span of time (not permanently) then, either the thread is in the blocked state or is in the waiting state. For example, a thread (A) may want to print some data from the printer. However, at the same time, the other thread (B) is using the printer to print some data. Therefore, thread A has to wait for thread B to use the printer. Thus, thread A is in the blocked state.

Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:**
3. **Blocked / Waiting:**
4. **Timed Waiting:** Sometimes, waiting for leads to starvation. For example, a thread (A) has entered the critical section of a code and is not willing to leave that critical section. In such a scenario, another thread (B) has to wait forever, which leads to starvation. To avoid such scenario, a timed waiting state is given to thread B. A real example of timed waiting is when we invoke the sleep(#Time) method on a specific thread.

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

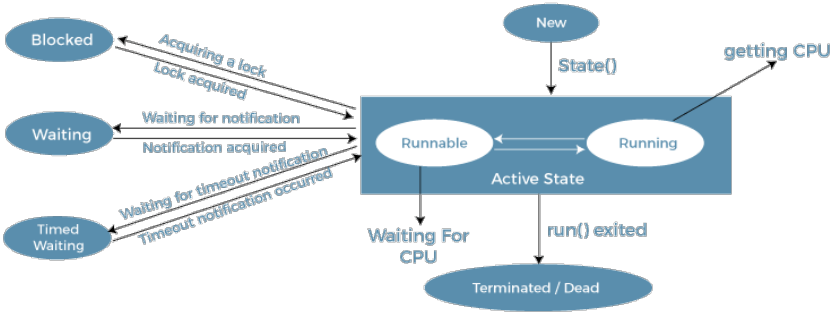
Life cycle of a Thread (Thread States)

In Java, a thread always exists in any one of the following states. These states are:

1. **New:**
2. **Active:**
3. **Blocked / Waiting:**
4. **Timed Waiting:**
5. **Terminated:** A thread reaches the termination state because of the following reasons:1) When a thread has finished its job, then it exists or terminates normally. 2) Abnormal termination: It occurs when some unusual events such as an unhandled exception or segmentation fault.

Life cycle of a Thread (Thread States)

The following diagram shows the different states involved in the life cycle of a thread.



Life Cycle of a Thread

Life cycle of a Thread (Thread States)

Homework: read and understand the example at Java Point (click me).

Create Java Threads

Method 1: Create a thread in Java by extending Thread class

For 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();
 }
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 1: Create a thread in Java by extending Thread class

For 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...

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 2: Create a thread in Java by implementing Runnable interface

For 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);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 2: Create a thread in Java by implementing Runnable interface

For 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);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

Output:

thread is running...

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 3: Create a thread in Java by using the Thread Class: Thread(String Name)

For example:

```
public class MyThread1
{
// Main method
public static void main(String argvs[])
{
// creating an object of the Thread class using the constructor Thread(String name)
Thread t= new Thread("My first thread");

// the start() method moves the thread to the active state
t.start();
// getting the thread name by invoking the getName() method
String str = t.getName();
System.out.println(str);
}
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 3: Create a thread in Java by using the Thread Class: Thread(String Name)

For example:

```
public class MyThread1
{
// Main method
public static void main(String argvs[])
{
// creating an object of the Thread class using the constructor Thread(String name)
Thread t= new Thread("My first thread");

// the start() method moves the thread to the active state
t.start();
// getting the thread name by invoking the getName() method
String str = t.getName();
System.out.println(str);
}
}
```

Output:

thread is running...

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 4: Create a thread in Java by using the Thread Class: Thread(Runnable r, String name)

For example:

```
// main method
public static void main(String argvs[])
{
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r, String name)
Thread th1 = new Thread(r1, "My new thread");

// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```

```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Create Java Threads

Method 4: Create a thread in Java by using the Thread Class: Thread(Runnable r, String name)

For example:

```
// main method
public static void main(String argvs[])
{
// creating an object of the class MyThread2
Runnable r1 = new MyThread2();

// creating an object of the class Thread using Thread(Runnable r, String name)
Thread th1 = new Thread(r1, "My new thread");

// the start() method moves the thread to the active state
th1.start();

// getting the thread name by invoking the getName() method
String str = th1.getName();
System.out.println(str);
}
}
```

Output:

My new thread

Now the thread is running ...

```
public class MyThread2 implements Runnable
{
public void run()
{
System.out.println("Now the thread is running ...");
}
```

Thread Scheduler in Java

A component of Java that decides which thread to run or execute and which thread to wait for is called a thread scheduler in Java. There are some criteria that decide which thread will execute first.

There are two factors for scheduling a thread:

- **Priority**: If a thread has a higher priority, it means that the thread has got a better chance of getting picked up by the thread scheduler.
- **Time of Arrival**: A thread that arrived first gets the preference over the other threads.

Thread Scheduler Algorithms

- **First Come First Serve Scheduling**: In this scheduling algorithm, the scheduler picks the threads that arrive first in the runnable queue.

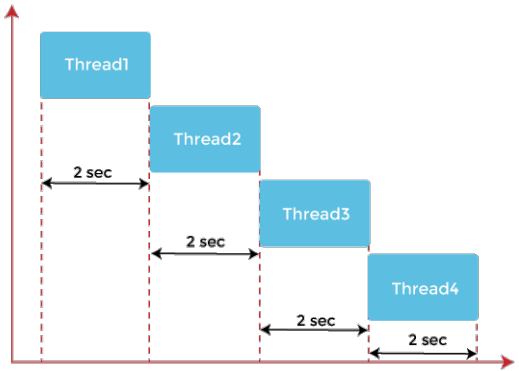
First Come First Serve Scheduling

- **Time-slicing scheduling**:
- **Preemptive-Priority Scheduling**:

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Thread Scheduler Algorithms

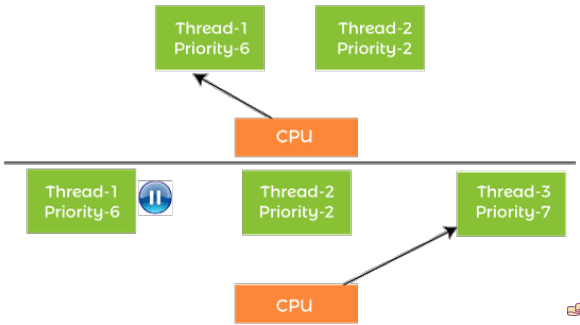
- **First Come First Serve Scheduling**:
- **Time-slicing scheduling**: some time-slices are provided to the threads so that after some time, the running thread has to give up the CPU.



Time slicing scheduling

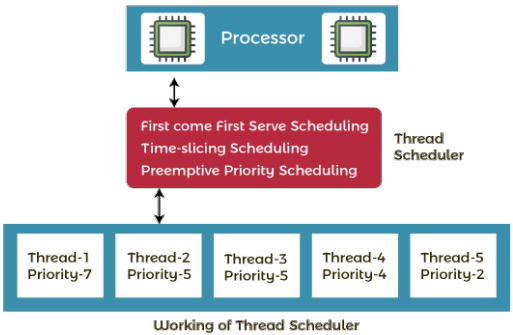
Thread Scheduler Algorithms

- **First Come First Serve Scheduling**:
- **Time-slicing scheduling**:
- **Preemptive-Priority Scheduling**: The name of the scheduling algorithm denotes that the algorithm is related to the priority of the threads.



Preemptive-Priority Scheduling

Working of the Java Thread Scheduler



Working of Thread Scheduler

- It is the responsibility of the thread scheduler to decide which thread will get the CPU.
- If another thread (that has higher priority) reaches in the runnable state, then the current thread is pre-empted from the processor, and the arrived thread with higher priority gets the CPU time.

Java sleep() Method

The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is the sleeping time.

```
class TestSleepMethod1 extends Thread{
 public void run(){
 for(int i=1;i<5;i++){
 // the thread will sleep for the 500 milli seconds
   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();
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java sleep() Method

The method sleep() is being used to halt the working of a thread for a given amount of time. The time up to which the thread remains in the sleeping state is the sleeping time.

```
class TestSleepMethod1 extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
  // the thread will sleep for the 500 milli seconds
    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();
```

Output:

1
1
2
2
3
3

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java join() method

The join() method permits one thread to wait until the other thread to finish its execution.

```
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();
 }
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java join() method

The join() method permits one thread to wait until the other thread to finish its execution.

```
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();
 }
```

Output:

```
1
2
3
1
1
2
2
3
3
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java setName() method

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. thread-0, thread-1 and so on.

```
class TestMultiNaming1 extends Thread{
 public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
  TestMultiNaming1 t1=new TestMultiNaming1();
  TestMultiNaming1 t2=new TestMultiNaming1();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());

  t1.start();
  t2.start();

  t1.setName("Sonoo Jaiswal");
  System.out.println("After changing name of t1:"+t1.getName());
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java setName() method

The Thread class provides methods to change and get the name of a thread. By default, each thread has a name, i.e. thread-0, thread-1 and so on.

```
class TestMultiNaming1 extends Thread{
 public void run(){
  System.out.println("running...");
 }
 public static void main(String args[]){
  TestMultiNaming1 t1=new TestMultiNaming1();
  TestMultiNaming1 t2=new TestMultiNaming1();
  System.out.println("Name of t1:"+t1.getName());
  System.out.println("Name of t2:"+t2.getName());

  t1.start();
  t2.start();

  t1.setName("Sonoo Jaiswal");
  System.out.println("After changing name of t1:"+t1.getName());
```

Output:

Name of t1:Thread-0

Name of t2:Thread-1

After changing name of t1:Sonoo Jaiswal

running...

running...

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java getPriority() and setPriority() methods

- The **Thread.getPriority()** method returns the priority of the given thread.
- The **Thread.setPriority()** method assign the priority of the thread to newPriority.

```
// the main method
public static void main(String argvs[])
{
// Creating threads with the help of ThreadPriorityExample class
ThreadPriorityExample th1 = new ThreadPriorityExample();
ThreadPriorityExample th2 = new ThreadPriorityExample();
ThreadPriorityExample th3 = new ThreadPriorityExample();
```

```
// Setting priorities of above threads by
// passing integer arguments
th1.setPriority(6);
th2.setPriority(3);
th3.setPriority(9);

// 6
System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 3
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 9
System.out.println("Priority of the thread th3 is : " + th3.getPriority());
```

Java getPriority() and setPriority() methods

- The **Thread.getPriority()** method returns the priority of the given thread.
- The **Thread.setPriority()** method assign the priority of the thread to newPriority.

```
// the main method
public static void main(String argvs[])
{
// Creating threads with the help of ThreadPriorityExample class
ThreadPriorityExample th1 = new ThreadPriorityExample();
ThreadPriorityExample th2 = new ThreadPriorityExample();
ThreadPriorityExample th3 = new ThreadPriorityExample();
```

```
// Setting priorities of above threads by
// passing integer arguments
th1.setPriority(6);
th2.setPriority(3);
th3.setPriority(9);

// 6
System.out.println("Priority of the thread th1 is : " + th1.getPriority());

// 3
System.out.println("Priority of the thread th2 is : " + th2.getPriority());

// 9
System.out.println("Priority of the thread th3 is : " + th3.getPriority());
```

Output:

Priority of the thread th1 is : 6

Priority of the thread th2 is : 3

Priority of the thread th3 is : 9

Java Thread Pool

Java Thread pool represents a group of worker threads that are waiting for the job and reused many times.

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
class WorkerThread implements Runnable {
   private String message;
   public WorkerThread(String s){
      this.message=s;
   }
    public void run() {
      System.out.println(Thread.currentThread().getName()+" (Start) message = "+message);
      processmessage();//call processmessage method that sleeps the thread for 2 seconds
      System.out.println(Thread.currentThread().getName()+" (End)");//prints thread name
   }
   private void processmessage() {
      try {  Thread.sleep(2000);  } catch (InterruptedException e) { e.printStackTrace(); }
   }
}
```

```
public class TestThreadPool {
   public static void main(String[] args) {
      ExecutorService executor = Executors.newFixedThreadPool(5);//creating a pool of 5 threads
      for (int i = 0; i < 10; i++) {
         Runnable worker = new WorkerThread("" + i);
         executor.execute(worker);//calling execute method of ExecutorService
       }
      executor.shutdown();
      while (!executor.isTerminated()) {   }

      System.out.println("Finished all threads");
   }
}
```

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

Java Thread Pool

Output:

pool-1-thread-1 (Start) message = 0
pool-1-thread-2 (Start) message = 1
pool-1-thread-3 (Start) message = 2
pool-1-thread-5 (Start) message = 4
pool-1-thread-4 (Start) message = 3
pool-1-thread-2 (End)
pool-1-thread-2 (Start) message = 5
pool-1-thread-1 (End)
pool-1-thread-1 (Start) message = 6
pool-1-thread-3 (End)
pool-1-thread-3 (Start) message = 7
pool-1-thread-4 (End)
pool-1-thread-4 (Start) message = 8
pool-1-thread-5 (End)
pool-1-thread-5 (Start) message = 9
pool-1-thread-2 (End)
pool-1-thread-1 (End)
pool-1-thread-4 (End)
pool-1-thread-3 (End)
pool-1-thread-5 (End)