420-B20

# Threads and Concurrent Programming

## What is a thread?

* a simple sequence of executable statements within a program
* an independent path of execution within a program.
* In a Java application, a thread begins at **main()** and continues sequentially through the program statements.

Visualize a thread: Imagine a list of the program's statements as they are executed by the CPU

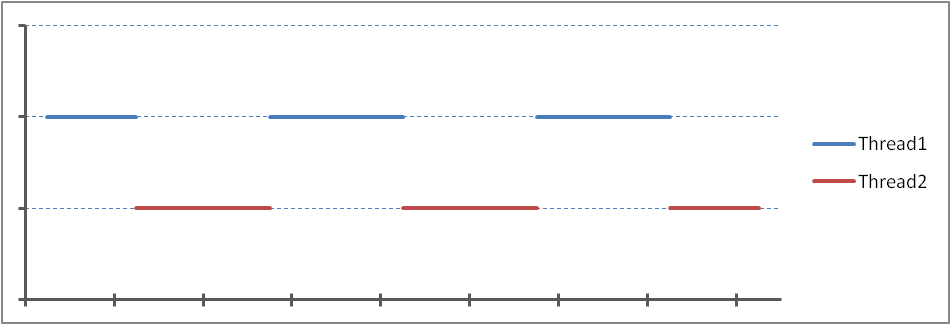
Now imagine breaking a program up into two or more independent threads:

* each thread has its own sequence of instructions
* Within a single thread, the statements are executed one after the other as usual.
* But by alternating statements from one thread with statements from another, the computer can run several threads concurrently.

### Multi-threading

* Multithreading refers to two or more tasks executing concurrently within a single program.
* The Java Virtual Machine is *multi-threaded* -- it has more than one thread executing. For example, the *Garbage collector* thread collects memory of discarded objects

Each Thread gets a slice of the CPU's time



## Creating Threads

Two methods:

1. Create a subclass of the **Java Thread** class
2. Implement **Runnable** in a class

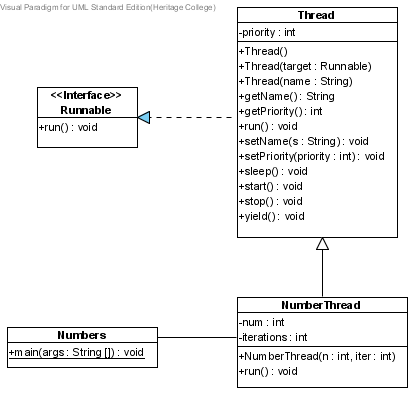
### Method 1. Creating a subclass of the Thread class

1. Create a subclass of the **Thread** class
2. Within the subclass, override the **Thread** **run()** method with the statements to be executed by the thread
3. Create several instances of the subclass and start each thread by invoking the **start()** method on each instance

Thread Creation

A thread can be created by creating a subclass of **Thread** and overriding the default **run()** method.

***Example:***

Create a **NumberThread** class that will display a number a given number of times. It should be able to run at the same time as another program.

***Class Diagram:***

***Java Code:***

// Create a subclass of the Thread class

public class NumberThread

extends Thread

{

private int num;

private int iterations;

public NumberThread(int n, int iter)

{

super();

num = n;

iterations = iter;

} // NumberThread(int, int)

// Override the Thread run() method with the statements

// to be executed by the thread

public void run()

{

for (int k= 0; k < iterations; ++k)

System.out.print(num);

System.out.println("\nEnd of NumberThread for " + num);

} // run()

} // NumberThread

Create a main class to test it:

public class Numbers {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.in);

NumberThread number1, number2, number3, number4;

System.out.println("How many iterations do you want?");

int iter = keyboard.nextInt();

// Create several instances of the subclass

number1 = new NumberThread(1, iter);

number2 = new NumberThread(2, iter);

number3 = new NumberThread(3, iter);

number4 = new NumberThread(4, iter);

// Start each thread by invoking the start() method

// on each instance

System.out.println("\nStarting number1 now.");

number1.start();

System.out.println("\nStarting number2 now.");

number2.start();

System.out.println("\nStarting number3 now.");

number3.start();

System.out.println("\nStarting number4 now.");

number4.start();

} // main()

} // Numbers

***Sample run***:

How many iterations do you want?

10

Starting number1 now.

Starting number2 now.

11111

Starting number3 now.

11111

Starting number4 now.

End of NumberThread for 1

3333333333

End of NumberThread for 3

4444444444

End of NumberThread for 4

2222222222

End of NumberThread for 2

### p707f1Operation:

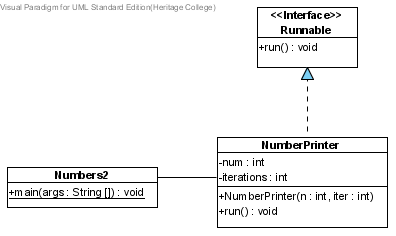
### Method 2: Implementing the Runnable interface

1. Implement the **Runnable** interface for an existing class and implement the **run()** method containing the statements to be executed by the thread.
2. Create several **Thread** instances by first creating an instance of the **Runnable** class and passing each instance as an argument to the **Thread()** constructor
3. Start each thread instance by invoking the **start()** method on it.

Thread Creation

A thread can be created by passing a **Runnable** object to a new **Thread** instance. The object's **run()** method will be invoked automatically as soon as the thread's **start()** method is called.

***The previous example using the Runnable Interface Method:***

***Class Diagram:***

***Java Code:***

// Implement the Runnable interface for an existing class

public class NumberPrinter

implements Runnable

{

private int num;

private int iterations;

public NumberPrinter(int n, int iter)

{

super();

num = n;

iterations = iter;

} // NumberPrinter(int, int)

// Implement the run() method containing the statements to

// be executed by the thread

public void run()

{

for (int k = 0; k < iterations; ++k)

System.out.print(num);

System.out.println("\nEnd of NumberThread for " + num);

} // run()

} // NumberPrinter

public class Numbers2

{

public static void main(String[] args)

{

Scanner keyboard = new Scanner(System.in);

// Create several Thread instances

Thread number1, number2, number3, number4;

System.out.println("How many iterations do you want?");

int iter = keyboard.nextInt();

// Create instances of the Runnable class passing each instance

// as an argument to the Thread() constructor

number1 = new Thread(new NumberPrinter(1, iter));

number2 = new Thread(new NumberPrinter(2, iter));

number3 = new Thread(new NumberPrinter(3, iter));

number4 = new Thread(new NumberPrinter(4, iter));

// Start each thread instance by invoking the start() method.

System.out.println("\nStarting number1 now.");

number1.start();

System.out.println("\nStarting number2 now.");

number2.start();

System.out.println("\nStarting number3 now.");

number3.start();

System.out.println("\nStarting number4 now.");

number4.start();

} // main()

} // Numbers2

***Sample run:***

How many iterations do you want?

20

Starting number1 now.

Starting number2 now.

11111

Starting number3 now.

111111

Starting number4 now.

111111111

End of NumberThread for 1

23333332343222222222222222222

End of NumberThread for 2

333333333333

End of NumberThread for 3

4444444444444444444

End of NumberThread for 4

***Exercise***:

Use the **Runnable** interface to convert the following class into a thread. You want the thread to print all the odd numbers up to its bound.

public class PrintOdds

{

private int bound;

public PrintOdds(int b)

{

super();

bound = b;

} // PrintOdds(int)

public void print()

{

System.out.print(name + " Odds to " + bound + " ");

for (int k = 1; k < bound; k += 2)

System.out.print(k + " ");

System.out.println(" End odds to " + bound);

} // print()

} // PrintOdds

***Solution:***

public class PrintOdds

implements Runnable

{

private int bound;

public PrintOdds(int b)

{

super();

bound = b;

}

public void run()

{

System.out.print(name + " Odds to " + bound + " ");

for (int k = 1; k < bound; k += 2)

System.out.print(k + " ");

System.out.println(" End odds to " + bound);

}

}

Write a class (TestPrintOdds) to run 3 threads of PrintOdds.

***Solution:***

public class TestPrintOdds

{

public static void main(String[] args)

{

Scanner keyboard = new Scanner(System.in);

Thread odd1, odd2, odd3;

System.out.println("What is the upper bound?");

int bound = keyboard.nextInt();

odd1 = new Thread(new PrintOdds(bound));

System.out.println("Starting odd1");

odd1.start();

odd2 = new Thread(new PrintOdds(bound));

System.out.println("Starting odd2");

odd2.start();

odd3 = new Thread(new PrintOdds(bound));

System.out.println("Starting odd3");

odd3.start();

} // main()

} // TestPrintOdds

## Thread Priority

- priority determines which thread will run ***next***

- ***higher*** priority threads run before (and longer than) ***lower*** priority threads

**- Thread.MAX\_PRIORITY**: defines the upper limit for the priority

**- Thread.MIN\_PRIORITY**: defines the lower limit for the priority

Pre-emption

A higher-priority thread that wants to run will pre-empt any threads of lower priority.

***Example:*** Modify **NumberThread** to set the priority to the number that is being displayed and alter the display loop so that it displays fewer numbers.

***NumberThread:***

public class NumberThread extends Thread

{

private int num, iterations;

public NumberThread(int n, int iter)

{

super();

num = n;

iterations = iter;

if (iterations < 1000000)

iterations \*= 1000000;

setPriority(num);

} // NumberThread(int, int)

public void run()

{

for (int k = 0; k < iterations; ++k) // loops *iteration* times

if (k % 1000000 == 0) // Only prints every millionth number

System.out.print(num);

System.out.println();

} // run()

} // NumberThread

***Sample run:***

Lowest priority: 1 Highest priority: 10

Starting number1 now. Priority 1

Starting number2 now. Priority 2

1

Starting number3 now. Priority 3

Starting number4 now. Priority 4

414141414

End of NumberThread for 1

End of NumberThread for 4

3

232323232

End of NumberThread for 3

End of NumberThread for 2

Platform Dependence

Thread implementation in Java is platform dependent. Adequate testing is necessary to ensure that a program will perform correctly on a given platform.

***Thread Coordination***

One way to coordinate the behaviour of two threads is to give one thread higher priority than another.

### Thread methods

**Thread.yield()**

* Causes the thread to yield the CPU, allowing the thread scheduler to choose another thread.
* The thread scheduler will reschedule it.

**Thread.sleep()**

* Causes the thread to yield the CPU for a given number of milliseconds, allowing the thread scheduler to choose another thread.
* It will not be rescheduled until a given time period has elapsed.
* throws an **InterruptedException**,which is checked and therefore must be caught

***Example:***

public void run()

{

for (int k = 0; k < iterations; ++k)

// loops iteration times

if (k % 1000000 == 0) // Only prints every millionth number

{

if (num % 2 == 1 && k % 5000000 == 0)

{

try

{

System.out.println("\nNumber " + num + " going to sleep");

Thread.sleep((long) (Math.random() \* 10000));

}

catch (InterruptedException e)

{

System.out.println(e.getMessage());

} // catch

} // if (num % 2 == 1 && k % 5000000 == 0)

System.out.print(num);

} // if (k % 1000000 == 0)

System.out.println();

}// run()

***Sample run:***

How many iterations do you want?

10

Starting number1 now.

Starting number2 now.

Number 1 going to sleep

Starting number3 now.

Starting number4 now.

Number 3 going to sleep

42424224242424242424

33333

Number 3 going to sleep

33333

11111

Number 1 going to sleep

11111

### Asynchronous Threads

* execution and time of a set of threads are unpredictable
* OS and JVM scheduling algorithms determine when a given thread will execute
* cannot determine when in the source code a thread might be pre-empted
* **Note**: even a simple arithmetic statement may be interrupted:

For example:

**int n = 5 + 3;**

translates to machine language code that performs the following steps:

**store 5 in register A**

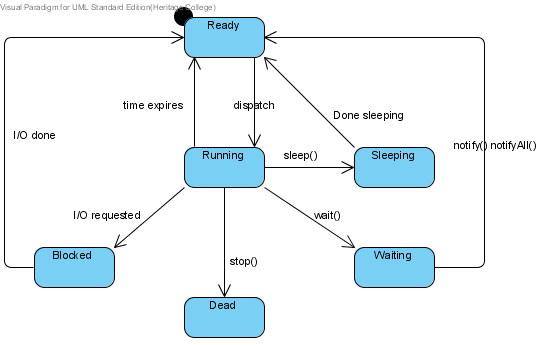
**add 3 to register A**

**store the value in register A in memory location N**

and may be interrupted between any two steps

### Thread States and Life Cycle

* every thread has a life cycle and may be in any one of 6 states:

***Summary of Thread States:***

|  |  |
| --- | --- |
| State | **Description** |
| Ready | The thread is ready to run and waiting for the CPU |
| Running | The thread is executing on the CPU |
| Waiting | The thread is waiting for some event to happen |
| Sleeping | The thread has been told to sleep for a time |
| Blocked | The thread is waiting for I/O to finish |
| Dead | The thread is terminated |