Introduction to Programming

Java - Multiple Inheritance and Multithreaded Programming

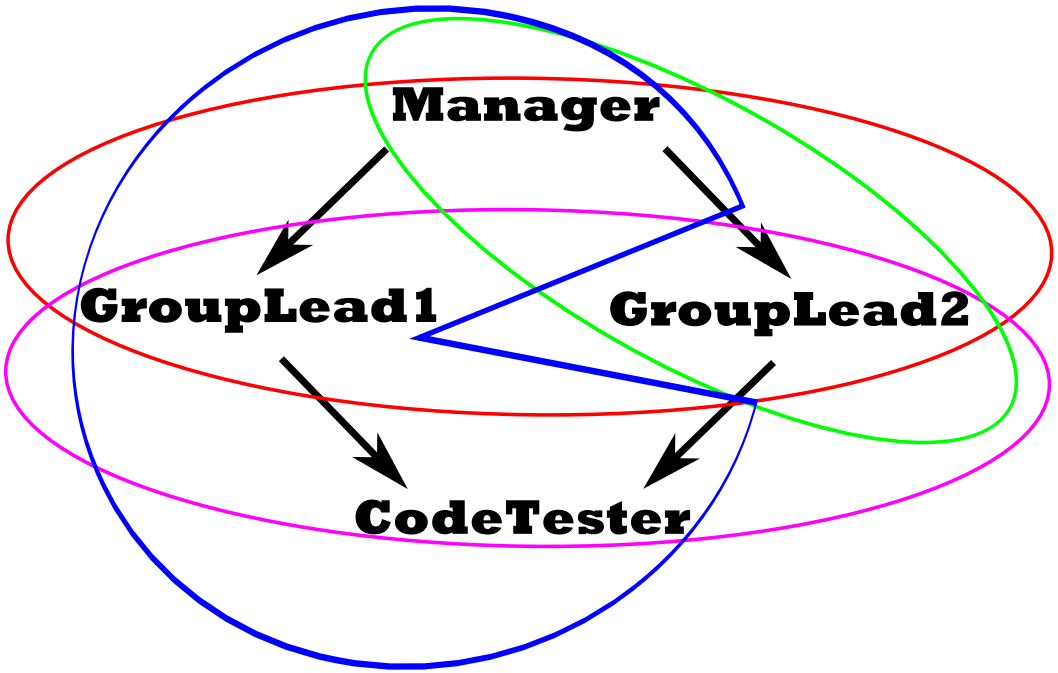
Malay Bhattacharyya

Assistant Professor

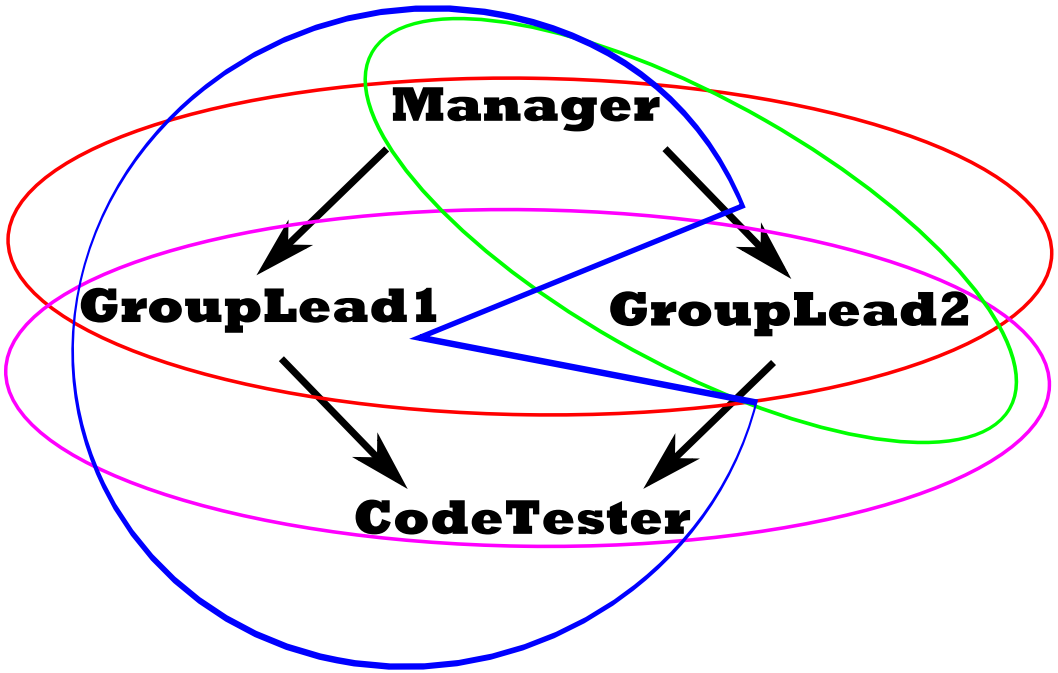
Machine Intelligence Unit
Indian Statistical Institute, Kolkata

February, 2021

Visualizing a complex form inheritance



Visualizing a complex form inheritance



All forms of inheritances together – single (shown in green), multi-level (shown in blue), hierarchical (shown in red) and multiple (shown in violet).

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

But, this is conceptually not possible in java

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

But, this is conceptually not possible in java

So what?

Why interfaces?

In multiple inheritance there exists more than one superclass for a single subclass

But, this is conceptually not possible in java

So what?

We use interfaces

Concept of interfaces

An interface is nothing but a class

Concept of interfaces

An interface is nothing but a class

The characteristics of interfaces:

- Interfaces are used as *superclasses* whose properties are inherited
- Interfaces support final fields and abstract methods

Defining an interface

```
interface IF{
    // Declaration of variables (always static final)
    ...
    // Declaration of methods (always abstract)
    ...
}
```

Extending interfaces

An interface (extends) an interface

Extending interfaces

An interface (extends) an interface

```
 interface IF1{
     ...
}
interface IF2{
     ...
}
interface subIF extends IF1, IF2{ ...  }
```

Extending interfaces

An interface (extends) an interface

```
 interface IF1{
     ...
}
interface IF2{
     ...
}
interface subIF extends IF1, IF2{ ...  }
```

Note: The extension of multiple interfaces gives you the power of implementing multiple inheritance

Implementing interfaces

A class (implements) an interface

Implementing interfaces

A class (implements) an interface

```
 interface IF{
     ...
}
class subClass implements IF{ ...  }
class subClass extends superClass implements IF{ ...
}
```

Implementing interfaces

A class (implements) an interface

```
 interface IF{
     ...
}
class subClass implements IF{ ...  }
class subClass extends superClass implements IF{ ...
}
```

Note: Implementing is nothing but extending an interface to a class

Implementing multiple inheritance

The concept:

Implementing multiple inheritance

The concept:

An interface can extend multiple interfaces

Implementing multiple inheritance

The concept:

An interface can extend multiple interfaces

A class can implement that interface extending multiple interfaces

Implementing multiple inheritance – An example

```
 interface IF1{
    int i = 1;
}
interface IF2{
    int j = 2;
}
interface IF3 extends IF1, IF2{
    int k = 3;
}
class Multiple implements IF3{
    public static void main(String args[]){
        System.out.println("In main()");
        System.out.println("In IF"+i);
        System.out.println("In IF"+j);
        System.out.println("In IF"+k);
}
```

Using methods from interfaces

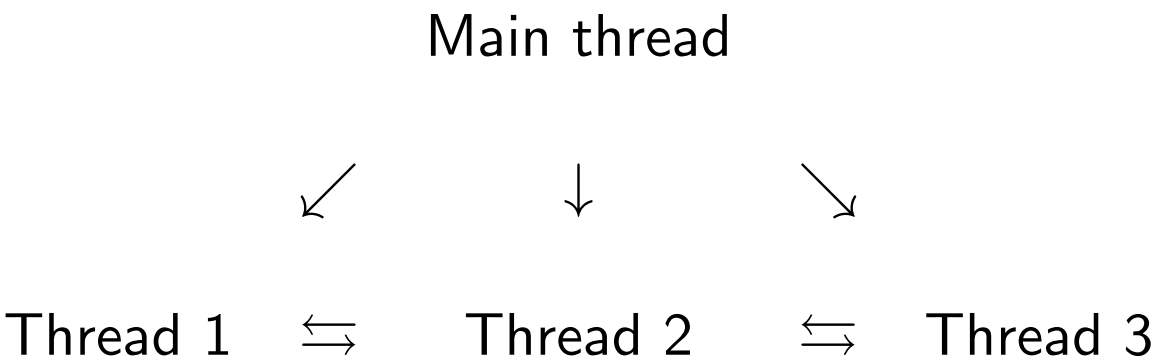
```
 interface IF{
    void display();
}
class Test implements IF{
    public void display(){
        System.out.println("Hi!");
    }
}
class UseIF{
    public static void main(String args[]){
        Test t = new Test();
        t.display();
    }
}
```

What is a thread?

A thread is a lightweight task

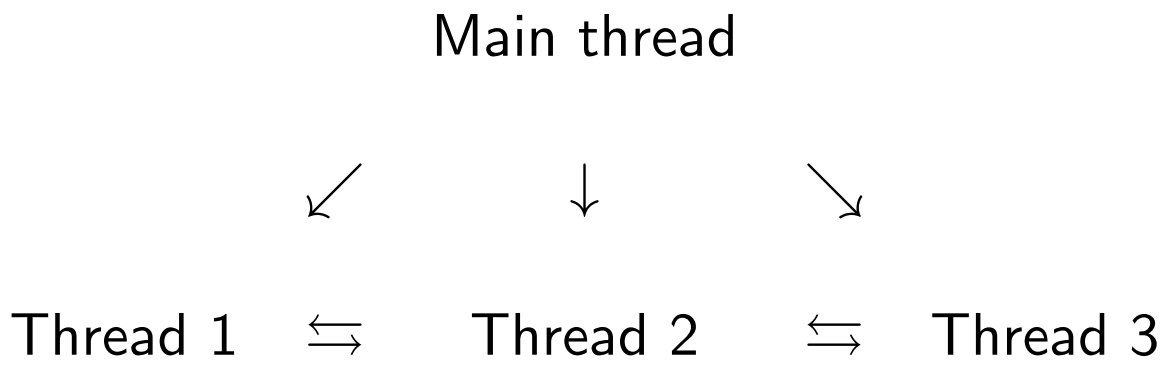
What is a thread?

A thread is a lightweight task



What is a thread?

A thread is a lightweight task



Note: Threads may switch or exchange data/results

What is a multithread?

A multithread is a combination of threads

What is a multithread?

A multithread is a combination of threads

Why multithreading?

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output
- True parallel execution on multiprocessor hardware

What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output
- True parallel execution on multiprocessor hardware
- Games with intelligent user agents and animation

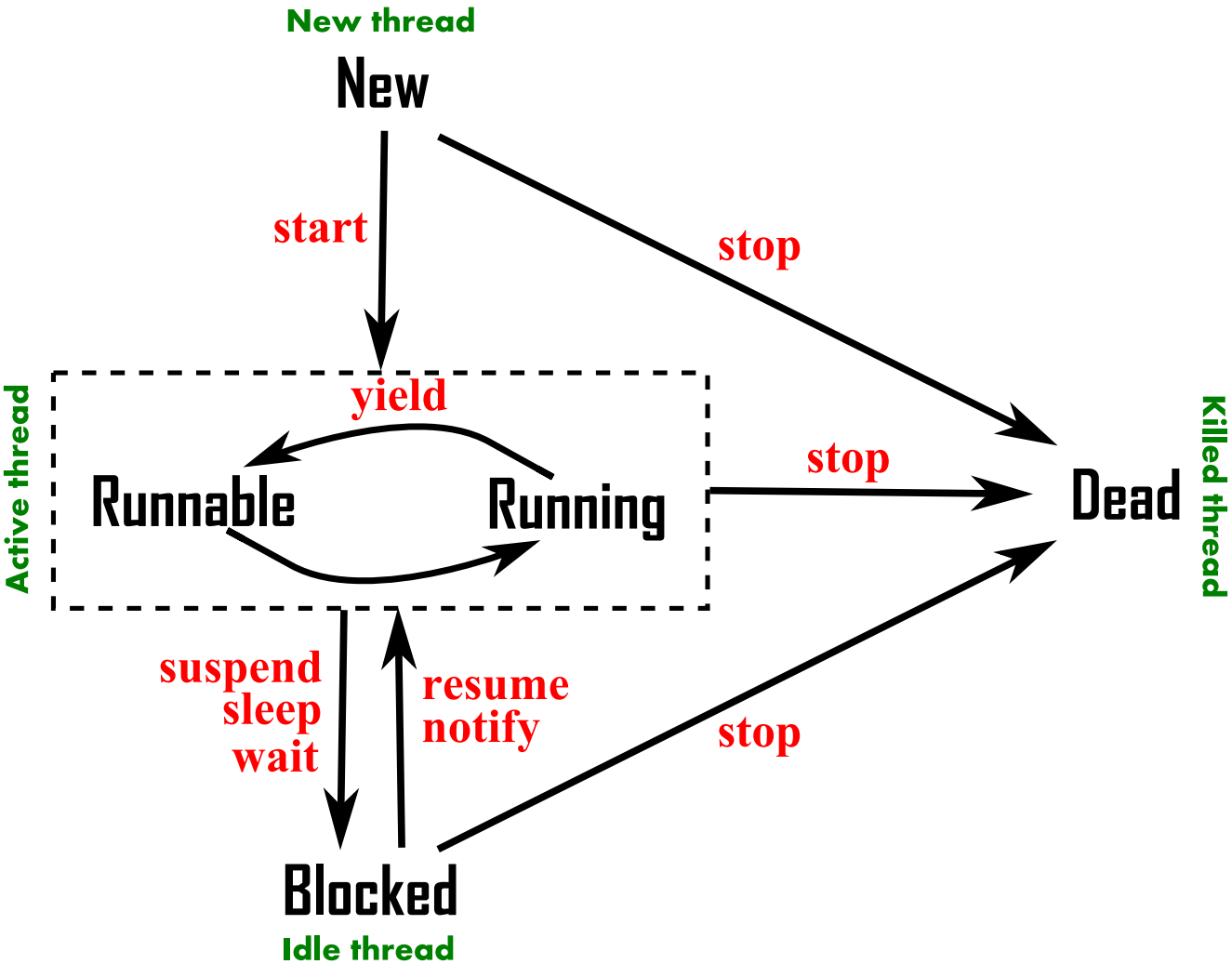
What is a multithread?

A multithread is a combination of threads

Why multithreading?

- To make sure that processors are fully utilized
- No block on input/output
- True parallel execution on multiprocessor hardware
- Games with intelligent user agents and animation
- Automatic garbage collection hidden from the user

Life cycle of a thread



State transition diagram

Creating a thread - using the Thread class

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
    }
}
```

Creating a thread - using the Thread class

```
class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
    }
}
```

Note: A thread should contain a `run()` method

Creating a thread - using the Runnable interface

```
class MyThread implements Runnable{
    public void run(){
        System.out.println("It is runnable ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        Thread t = new Thread(mt); // Initialization
        t.start(); // Start a thread
    }
}
```

Creating a thread - using the Runnable interface

```
class MyThread implements Runnable{
    public void run(){
        System.out.println("It is runnable ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        Thread t = new Thread(mt); // Initialization
        t.start(); // Start a thread
    }
}
```

Note: A thread is indirectly created with runnable interface

Multithreading in java

```
  class MyThread1 extends Thread{
    public void run(){
        for(int i=0;i<20;i++)
            System.out.println("Iteration "+i+" of 1 ...");
    }
}
class MyThread2 extends Thread{
    public void run(){
        for(int i=0;i<20;i++)
            System.out.println("Iteration "+i+" of 2 ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        new MyThread1().start(); // Start without declaration
        new MyThread2().start(); // Start without declaration
    }
}
```

Killing a thread

```
 class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
        mt.stop(); // Stop a thread
    }
}
```

Blocking a thread

```
class MyThread extends Thread{
  public void run(){
      for(i=0;i<1000;i++){
          System.out.println("It is running ...");
          if(i==100){sleep(100);} // Blocked for 100ms
          wait(); // Blocked until notified
          notify(); // Notify and start
      }
  }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread(); // Declaration
        mt.start(); // Start a thread
        mt.suspend(); // Blocked until resumed
        mt.resume(); // Resume and start
    }
}
```

Priority of threads

```
class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread();
        mt.setPriority(3); // Prioritization
    }
}
```

Priority of threads

```
class MyThread extends Thread{
    public void run(){
        System.out.println("It is running ...");
    }
}
class ThreadExample{
    public static void main(String [] args){
        MyThread mt = new MyThread();
        mt.setPriority(3); // Prioritization
    }
}
```

Note: The `setPriority()` method can take arguments between 1 (minimum) to 10 (maximum) to quantify priorities where a normal priority is quantified by 5 (default).

Priority of threads – Illustrating with an example

```
  class MyThread1 extends Thread{
    public void run(){
        System.out.println("Entered in MyThread1 ...");
        for(int i=1;i<=10;i++) System.out.print("[MT1:  "+i+"]");
        System.out.println("Exit from MyThread1 ...");
} }
class MyThread2 extends Thread{
    public void run(){
        System.out.println("Entered in MyThread2 ...");
        for(int i=1;i<=10;i++) System.out.print("[MT2:  "+i+"]");
        System.out.println("Exit from MyThread2 ...");
} }
class MyThread3 extends Thread{
    public void run(){
        System.out.println("Entered in MyThread3 ...");
        for(int i=1;i<=10;i++) System.out.print("[MT3:  "+i+"]");
        System.out.println("Exit from MyThread3 ...");
} }
```

Priority of threads – Illustrating with an example

```
class ThreadExample{
  public static void main(String [] args){
      MyThread1 mt1 = new MyThread1();
      MyThread2 mt2 = new MyThread2();
      MyThread3 mt3 = new MyThread3();
      mt3.setPriority(Thread.MAX_PRIORITY);
      mt2.setPriority(MyThread1.getPriority()+1);
      mt1.setPriority(Thread.MIN_PRIORITY);
      System.out.println("Start MyThread1");
      mt1.start();
      System.out.println("Start MyThread2");
      mt2.start();
      System.out.println("Start MyThread3");
      mt3.start();
      System.out.println("End of the game!!!");
  }
}
```

Thread exceptions

- **ThreadDeath:** Thread has been killed
- **InterruptedException:** It cannot be handled in the current state
- **IllegalArgumentException:** Illegal method argument
- **Exception:** Any other types of exceptions

Thread exceptions

- **ThreadDeath:** Thread has been killed
- **InterruptedException:** It cannot be handled in the current state
- **IllegalArgumentException:** Illegal method argument
- **Exception:** Any other types of exceptions

Note: These exceptions can be verified by using as arguments with try-catch.