

## Multithreaded Programming in Java

Supta Richard Philip

[richard@aiub.edu](mailto:richard@aiub.edu)

---

### Multitasking and Multithread

- In computing, multitasking is a concept of performing multiple tasks (also known as processes) over a certain period of time by executing them concurrently, e.g. OS.
- A thread is a single sequence of execution within a program.
- Multithreading refers to multiple threads of control within single program.
- An executing instance of a program is called a process. Processes have their own address space and threads share the address space of process.
- There are two ways to create a thread:

By extending Thread class

By implementing Runnable interface.

- Both cases implement run method.
- Class without Thread

```
class Hi {
    public void show() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Hi");
        }
    }
}

class Hello {
    public void show() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Hello");
        }
    }
}

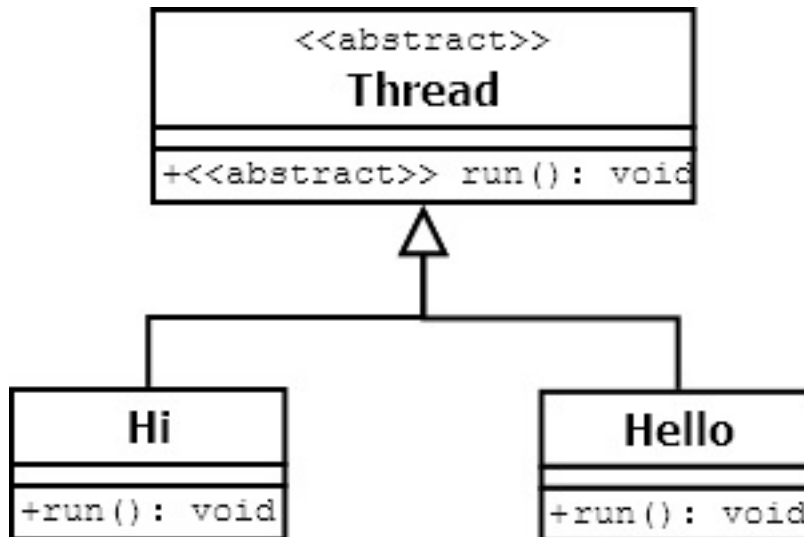
public class App3 {
    public static void main(String[] args) {
        Hi obj1 = new Hi();
        Hello obj2 = new Hello();
        obj1.show();
    }
}
```

```

        obj2.show();
    }
}

```

- Class with Thread

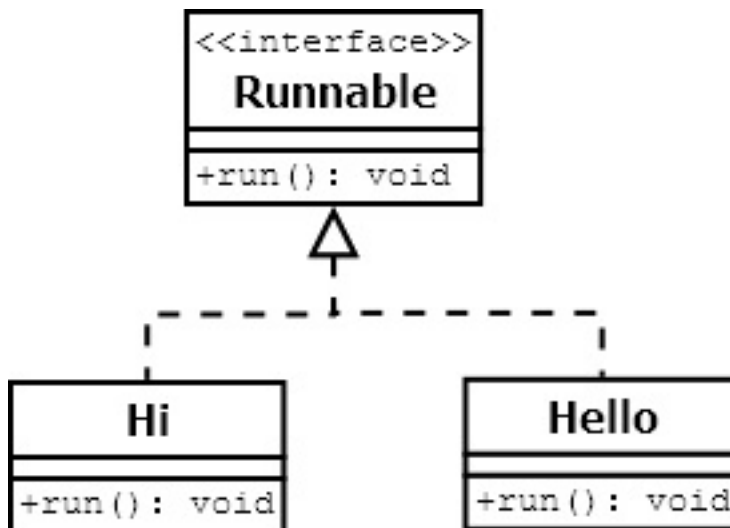


```

class Hi extends Thread{
    public void run() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Hi");
            try{Thread.sleep(1000);}catch(Exception e){}
        }
    }
}
class Hello extends Thread{
    public void run() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Hello");
            try{Thread.sleep(2000);}catch(Exception e){}
        }
    }
}
public class App3 {
    public static void main(String[] args) {
        Hi obj1 = new Hi();
        Hello obj2 = new Hello();
        obj1.start();
        obj2.start();
    }
}

```

- Thread implements Runnable interface



```
class Hi implements Runnable{
    public void run() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Hi");
            try{Thread.sleep(2000);}catch(Exception e){}
        }
    }
}

class Hello implements Runnable{
    public void run() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Hello");
            try{Thread.sleep(500);}catch(Exception e){}
        }
    }
}

public class App3 {
    public static void main(String[] args) {
        Hi obj1 = new Hi();
        Hello obj2 = new Hello();
        Thread t1 = new Thread(obj1);
        Thread t2 = new Thread(obj2);
        t1.start();
        t2.start();
    }
}
```

- More Example

```
//extends Thread Class---way1
class Myclass1 extends Thread {
    @Override
    public void run() {
        for (int i = 0; i < 10; i++) {
```

```

        System.out.println(Thread.currentThread().getId()+" Value " + i);
    }
    try {
        Thread.sleep(2000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
//Implements Runnable interface--way2
class Myclass2 implements Runnable {
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println(Thread.currentThread().getId()+" Value " + i);
        }
    }
}

class App {
    public static void main(String args[]) {
        //way1 test
        //Myclass1 c1 = new Myclass1();
        //c1.start();
        //Myclass1 c2 = new Myclass1();
        //c2.start();

        //way2 test
        Thread t1 = new Thread(new Myclass2());
        Thread t2 = new Thread(new Myclass2());
        t1.start();
        t2.start();
    }
}

```

- Creating an anonymous thread by implementing the Runnable interface.

```

public class App2 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 0; i < 10; i++) {
                    System.out.println(Thread.currentThread().getId() + " Value " +
                }
            }
        });
    }
}

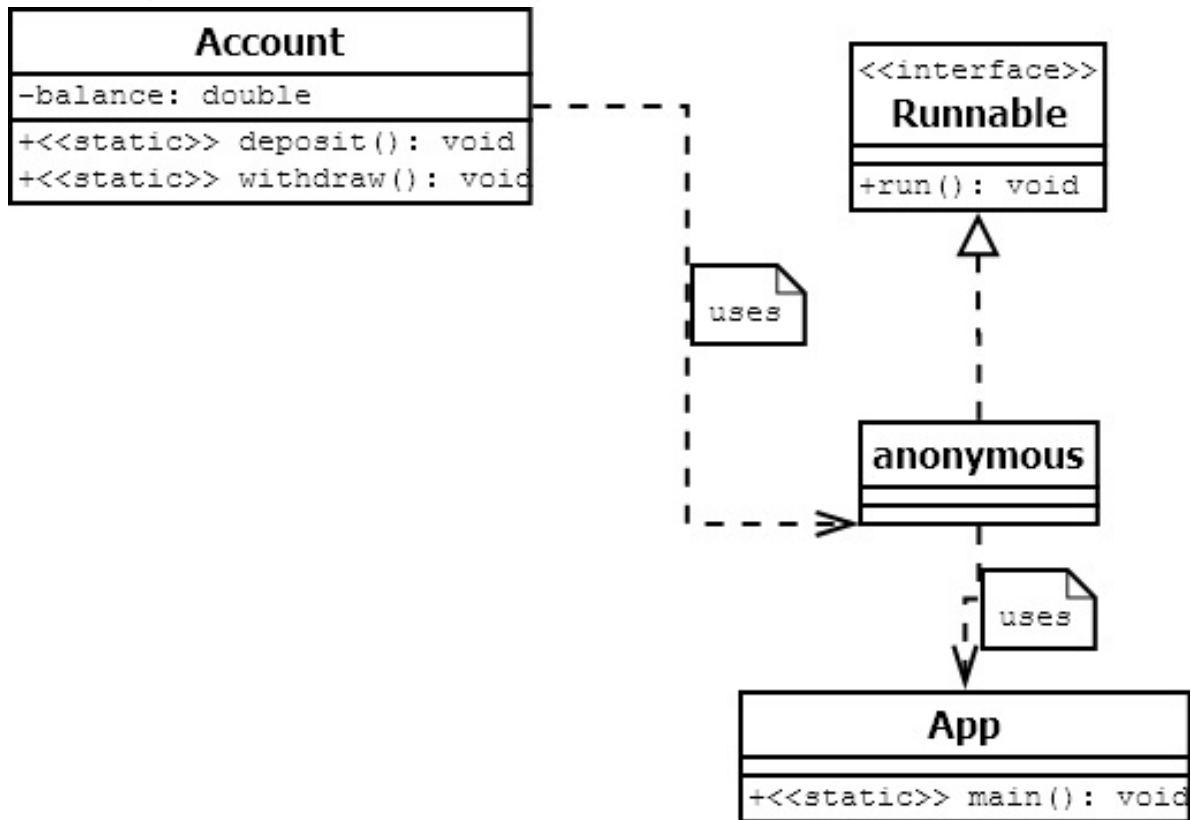
```

```

        t1.start();
    }
}

```

- Join and Synchronized(For Thread Safety)



```

class Account {
    private static double balance;
    public static synchronized void deposit() {
        balance = balance + 5;
    }
    public static synchronized void withdraw() {
        balance = balance - 1;
    }
    public static double getBalance() {
        return balance;
    }
}

public class App4 {
    public static void main(String[] args) {
        Thread t1 = new Thread(new Runnable() {
            @Override
            public void run() {
                for (int i = 1; i <= 100; i++) {
                    Account.deposit();
                }
            }
        })
    }
}

```

```
});  
Thread t2 = new Thread(new Runnable() {  
    @Override  
    public void run() {  
        for (int i = 1; i <= 100; i++) {  
            Account.withdraw();  
        }  
    }  
});  
t1.start();  
t2.start();  
try {  
    t1.join();  
    t2.join();  
} catch (InterruptedException e) {  
    e.printStackTrace();  
}  
System.out.println(Account.getBalance());  
}
```

## References

---

<https://www.youtube.com/playlist?list=PLzS3AYzXBoj8jq5o5qoIE4Kki19j1UrS8>

<https://www.youtube.com/playlist?list=PLmCsXDGbJHdjmXF-w2mWaXa5p0dxHE3nW>