# JAVA CONCURRENCY BASICS

- Extending *Thread* class

```java
public class HelloThread extends Thread {
    @Override
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        Thread thread = new HelloThread();
        thread.start();
    }
}
```

- Implementing *Runnable* interface

```java
public class HelloRunnable implements Runnable {
    @Override
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        Thread thread = new Thread(new HelloRunnable());
        thread.start();
    }
}
```
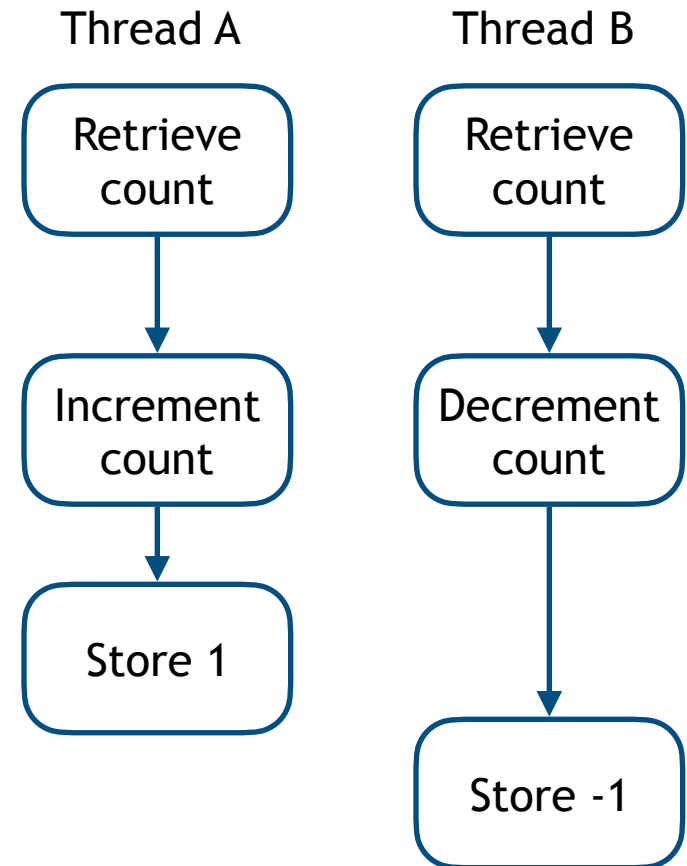
Which method is the best?
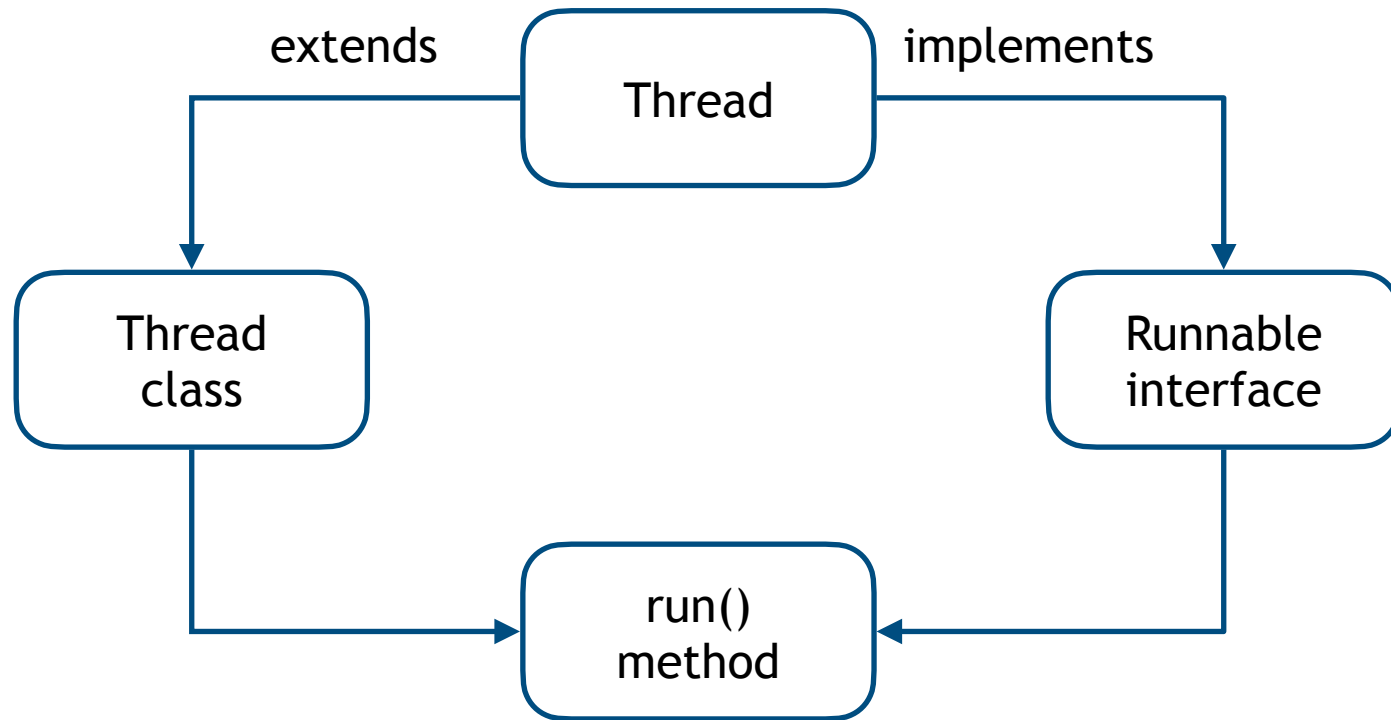
Which method is the best?

	Extend Thread	Implement Runnable
Inheritance option	no	yes
Loosely coupled	no	yes
Code reusability	no	yes
Method overhead	yes	no
Core reusability	no	yes

```
class Counter {
    private int count;

    public void increment() {
        count++;
    }

    public void decrement() {
        count--;
    }
}
```

Thread A

Retrieve count

Increment count

Store 1

Thread B

Retrieve count

Decrement count

Store -1

A's result is lost!

```
class SynchronizedCounter {
    private int count;

    public synchronized void increment() {
        count++;
     }


    public synchronized void decrement() {
        count--;
     }
}
```

```
class SyncBlockCounter {
    private int count;

    public void increment() {
        synchronized (this) {
            count++;
        }
    }

    public void decrement() {
        synchronized (this) {
            count--;
        }
    }
}
```

```java
public class DoubleCounter {
    private int count;
    private Object lock = new Object();

    public void increment() {
        synchronized(lock) {
            count++;
        }
    }

    public void decrement() {
        synchronized(lock) {
            count--;
        }
    }
}
```

```java
public class DoubleCounter {
    private int count1;
    private int count2;
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void increment1() {
        synchronized(lock1) {
            count1++;
        }
    }

    public void increment2() {
        synchronized(lock2) {
            count2++;
        }
    }
}
```

Which method is the best?
It **depends** on your problem!

- Advantages of synchronized methods:
  - More compact syntax
  - Enforces modularity

- Advantages of synchronized statements:
  - Flexibility
  - Reduced scope of lock
  - Better performance

- Pause execution

  ```
  sleep(milliseconds);
  ```

- Suspend a thread

  ```
  wait();
  ```

- Wake a thread up

  ```
  notify();
  ```

- Check if thread is still running

  ```
  isAlive();
  ```

- Temporary interrupt thread

  ```
  yield();
  ```

- Wait for another thread

  ```
  join();
  ```

- Stop thread

  ```
  stop();
  ```

1. Write a thread class that calls the `increment()` method of the class `Counter` for a given number of times. Create several threads, start them all, and wait for all the threads to terminate. Print the final value of the counter, and see whether it is correct.

2. Write a `synchronizedIncrement()` method in the class Counter, repeat the previous tests using the new method and compare the results.

3. Modify the `synchronizedIncrement()` method to use synchronized statements, repeat the previous tests using the new method and compare the results.

4. Write a program that uses multiple threads to find the integer with the largest number of divisors in a given range between `x` and `y`. Output the elapsed time, the integer that has the largest number of divisors, and the number of divisors that it has.