Java Threads

‹ Previous
Next ›

Java Threads

Threads allows a program to operate more efficiently by doing multiple things at the same time.

Threads can be used to perform complicated tasks in the background without interrupting the main program.

Creating a Thread

There are two ways to create a thread.

It can be created by extending the `Thread` class and overriding its `run()` method:

Extend Syntax

```
public class Main extends Thread {
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Another way to create a thread is to implement the `Runnable` interface:

Implement Syntax

```
public class Main implements Runnable {
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Running Threads

If the class extends the `Thread` class, the thread can be run by creating an instance of the class and call its `start()` method:

Extend Example

```
public class Main extends Thread {
  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    System.out.println("This code is outside of the thread");
  }
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Try it Yourself »

If the class implements the `Runnable` interface, the thread can be run by passing an instance of the class to a `Thread` object's constructor and then calling the thread's `start()` method:

Implement Example

```
public class Main implements Runnable {
  public static void main(String[] args) {
    Main obj = new Main();
    Thread thread = new Thread(obj);
    thread.start();
    System.out.println("This code is outside of the thread");
  }
  public void run() {
    System.out.println("This code is running in a thread");
  }
}
```

Try it Yourself »

Differences between "extending" and "implementing" Threads

The major difference is that when a class extends the Thread class, you cannot extend any other class, but by implementing the Runnable interface, it is possible to extend from another class as well, like: class `MyClass extends OtherClass implements Runnable`.

Concurrency Problems

Because threads run at the same time as other parts of the program, there is no way to know in which order the code will run. When the threads and main program are reading and writing the same variables, the values are unpredictable. The problems that result from this are called concurrency problems.

Example

A code example where the value of the variable **amount** is unpredictable:

```
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    System.out.println(amount);
    amount++;
    System.out.println(amount);
  }

  public void run() {
    amount++;
  }
}
```

Try it Yourself »

To avoid concurrency problems, it is best to share as few attributes between threads as possible. If attributes need to be shared, one possible solution is to use the `isAlive()` method of the thread to check whether the thread has finished running before using any attributes that the thread can change.

Example

Use `isAlive()` to prevent concurrency problems:

```
public class Main extends Thread {
  public static int amount = 0;

  public static void main(String[] args) {
    Main thread = new Main();
    thread.start();
    // Wait for the thread to finish
    while(thread.isAlive()) {
```

```
      System.out.println("Waiting...");
    }
    // Update amount and print its value
    System.out.println("Main: " + amount);
    amount++;
    System.out.println("Main: " + amount);
    }
    public void run() {
      amount++;
    }
}
```

Try it Yourself »

‹ Previous

Next ›

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References


3
schools