Application Design Using Java

Lecture 12

Debugging Tips I

- Print or log the value of any variable
  - `System.out.println("x=" + x);`
  - `Logger.getGlobal().info("x=" + x);`
- Put a separate main method in each class
- Use JUnit for unit testing
- Use a logging proxy

```
Random generator = new Random() {
  public double nextDouble() {
    double result = super.nextDouble();
    Logger.getGlobal().info("nextDouble: " + result);
    return result;
  }
};
```

- Get a stack trace

  -
    ```
    try
    {
      . . .
    }
    catch (Throwable t)
    {
      t.printStackTrace();
      throw t;
    }
    ```
  - `Thread.dumpStack();`

Debugging Tips II

- Redirect output to capture the errors

```
java MyProgram 1> errors.txt 2>&1
```

- Capture stack trace into a string

```
StringWriter out = new StringWriter();
new Throwable().printStackTrace(new PrintWriter(out));
String description = out.toString();
```

- Log stack traces into a file

```
Thread.setDefaultUncaughtExceptionHandler(
new Thread.UncaughtExceptionHandler() {
  public void uncaughtException(Thread t, Throwable e) {
    // save information in log file
  };
});
```

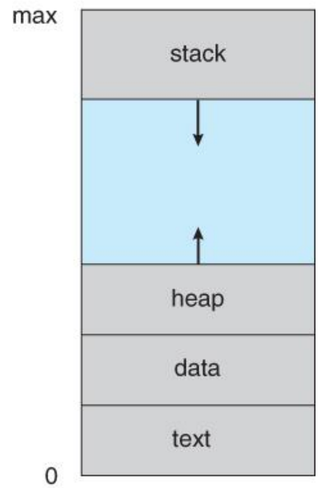
- Use -verbose flag with java command to watch class loading

Debugging Tips III

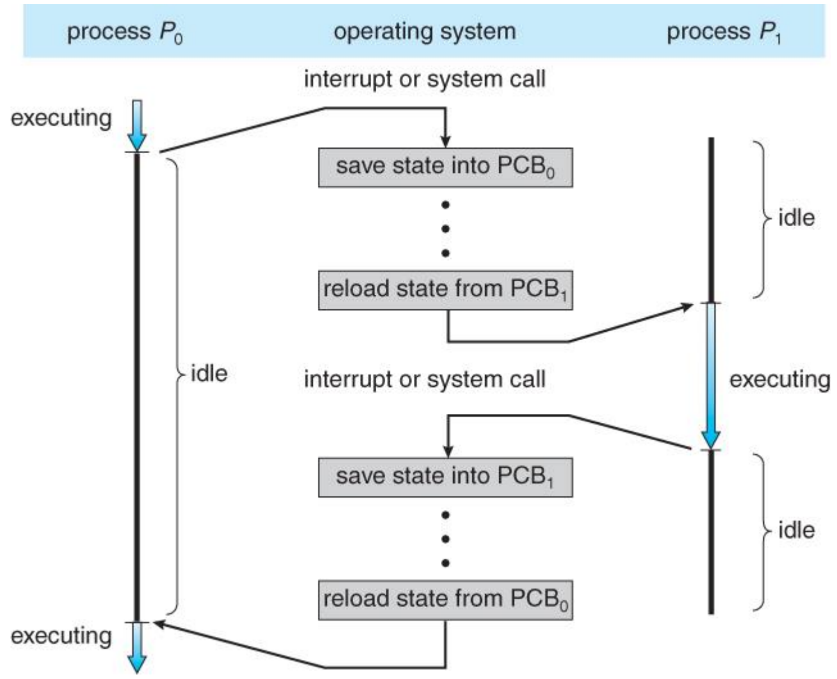
- Use -Xlint flag with javac command to spot common code problems
    - -Xlint:all
    - -Xlint:fallthrough
- Use Java Monitoring and Management Console (jconsole command)

Processes and Threads

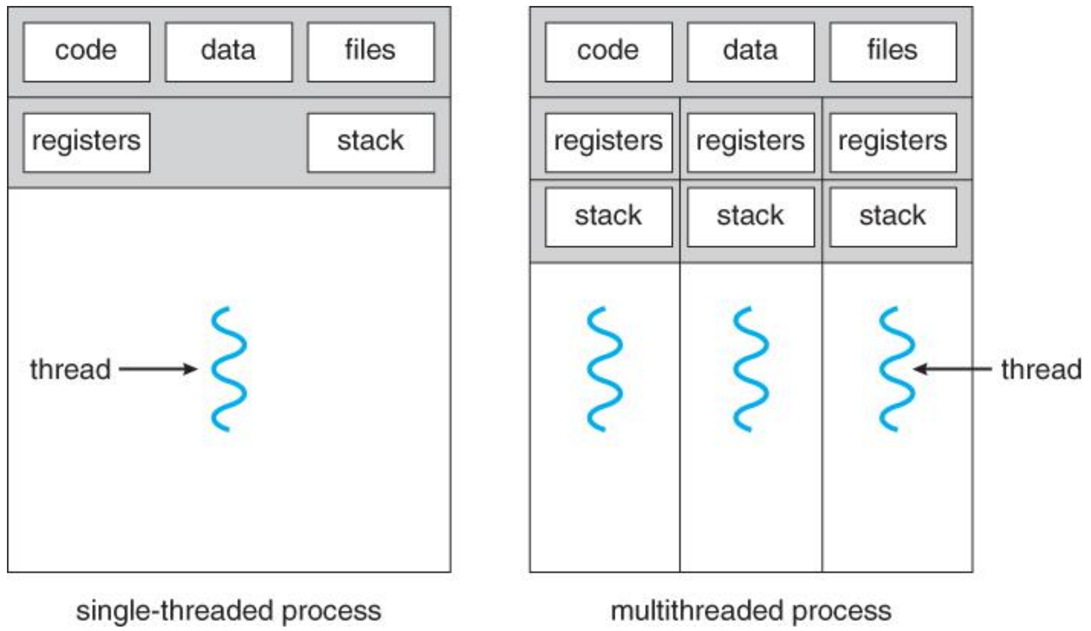
- run() method: the code to run as a new thread
- start() method: to actually start the thread



A process



Switching processes



Threads (lightweight processes)

Creating a thread I

- Extending Thread

```
public class T1 extends Thread {
  public void run(){
    ...
  }
  ...
}
Thread thread1= new T1();
thread1.start();
```

Creating a thread II

- Implementing Runnable

```
public class R1 implements Runnable{
  public void run(){
    ...
  }

   ...
}
Thread thread1 = new Thread(new R1());
thread1.start();
```

- Can be an anonymous class

```
new Thread(new Runnable() {
  public void run() {

    ...
  }
}).start();
```

Interrupting Threads

- Once started, a thread runs on its own
- We largely can’t force it terminate but we can request termination by sending thread an interrupt
- The interrupted thread can decide how to react to the interruption
- Typical thread code organization

```
Runnable r = new Runnable() {
  public void run() {
    try {
      . . .
      while (!Thread.currentThread().isInterrupted() && more work to do) {
        do more work
      }
    }
    catch(InterruptedException e) {
      // thread was interrupted during sleep or wait
    }
    finally {
      cleanup, if required
    }
    // exiting the run method terminates the thread
  }
};
```

//TODO before next lecture:

- Article Review and Presentation was posted. You need to let me know which article you want to review and get my approval. This ensures that there are no two students reviewing the same article. Drafts are due on 3/29 at 11:59 pm EDT. Must be submitted on Submitty.
- Java puzzler (posted on Submitty Forum)