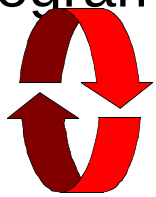
# SFDV3006
# Concurrent Programming

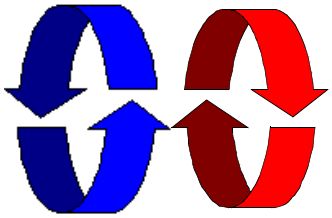
*Introduction and Java Threads*

# Concurrent Programming

- A sequential program has a single thread of control



- A concurrent program has multiple threads of control allowing it perform multiple computation in parallel and control multiple (external) activities which happen at the same time



# Why concurrent programming?

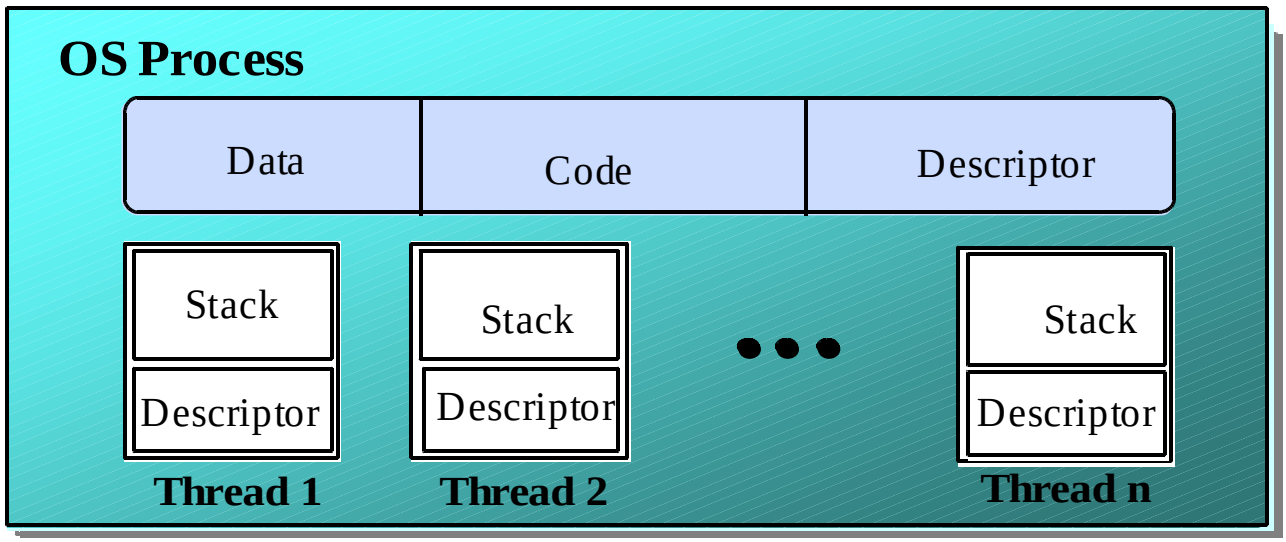
- Performance gain from multiprocessing hardware such as multiple CPUs, dual core CPUs
  - Get the advantage of parallel processing
- Increased application throughput
  - An I/O call need only block one thread
- Increased application responsiveness
  - High priority threads for user requests
- More appropriate structure for some programs
  - For example for programs which interact with environment, control multiple activities and handle multiple events (for example web servers)



# Issues in concurrency

- Software engineering /architecture
  - Identifying and implementing good primitives
- Performance
  - Maximising concurrency
- Correctness
  - Testing is difficult (because of non-deterministic behaviour)
  - Model – checking
  - Proof
  - In this course we will use FSP to check some models and prove that they are

# Implementing concurrent programs

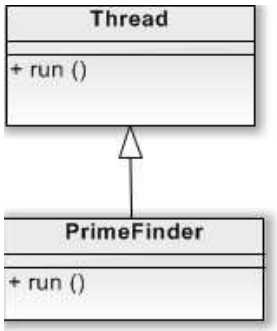
- Threads are commonly used for implementing concurrent programs
- 


Operating system view of threads – A heavyweight process in an operating system is represented by its code, data and the state of machine registers, given in a descriptor In order to support multiple threads a process has multiple stacks one for each thread. Threads are also called as lightweight processes (*Magee and Kramer*)

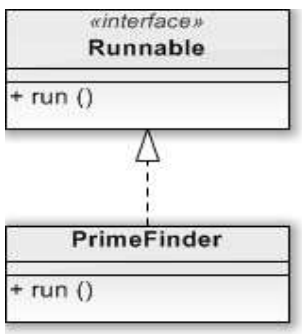


# Threads in Java

- From the beginning Java had support for creating and managing threads
- To create a thread in Java you need to either subclass from *java.lang.Thread* class or implement the *Runnable* interface which is also from the *java.lang* package.




Extending the *Thread* class




Implementing the *Runnable* interface

# Java and threads

- A good example of threads is the Java language itself
- Even if you create a program without using threads but with *main()* only – it still runs as a thread – the main thread
- Java also uses threads for various tasks such as garbage collection, object finalization and various other JVM housekeeping tasks
- Threads are used in JDK APIs such as Swing and AWT
- On the server side threads are commonly used in Servlet containers, Application Servers and RMI (Remote Method Invocation) for performance and scalability. These are most common Java programs used in the industry.
  - For example BlackBoard runs on a Servlet Container called Tomcat which is written in Java and uses threads heavily



## Creating your thread class– using the Thread class

- To create a thread using the *Thread* class – extend from the thread class and override the run method
- Example:

```
public class PrimeFinder extends Thread {
   public void run() {
      //code to find and print primes
   }
}
```

# Creating threads using Runnable

- To create a Thread using *Runnable* – implement the *Runnable* interface and override the *run()* method
- Code example:

```
public class PrimeFinder implements Runnable {
   public void run() {
      //code to find and print primes
   }
}
```



# Create the thread

- Before we can run the thread we need to create an instance of it and start it using the *start()* method which is inherited from the *Thread* class
- For starting the PrimeFinder

```
//somewhere is your main program
Thread primeFinderThread = new PrimeFinder();
primeFinderThread.start();
```

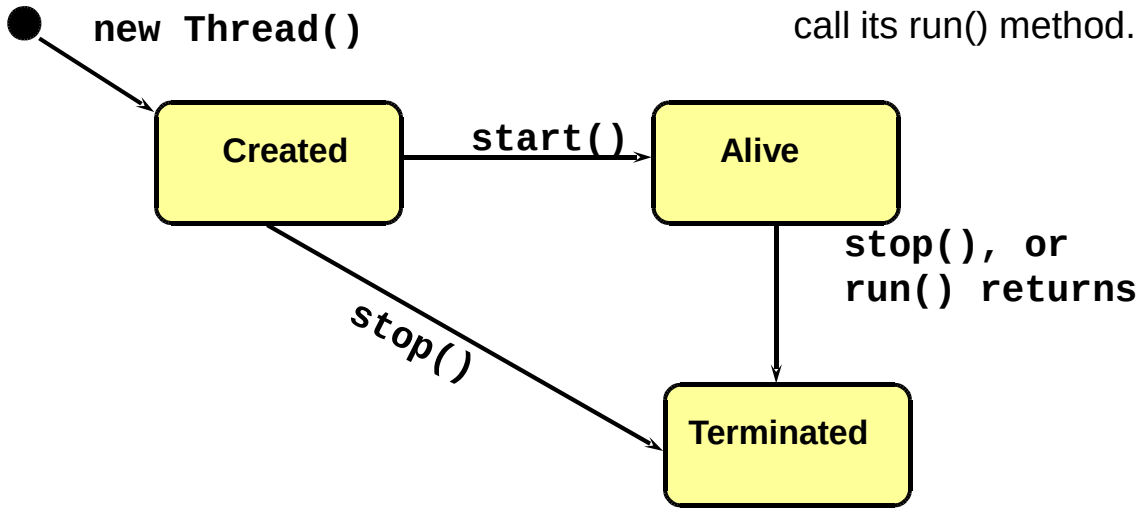
- Similarly for PrimeFinder implemented using Runnable

```
Thread primeFinder = new Thread(new PrimeFinder());
primeFinder.start();
```

# States of a thread - 1

- Once the thread is started it will have transitions between different states

**start()** causes the thread to call its run() method.




Note: methods like `stop()` are *deprecated* and should **NOT** be used. You will need refer to the Java API to know what other methods are deprecated.

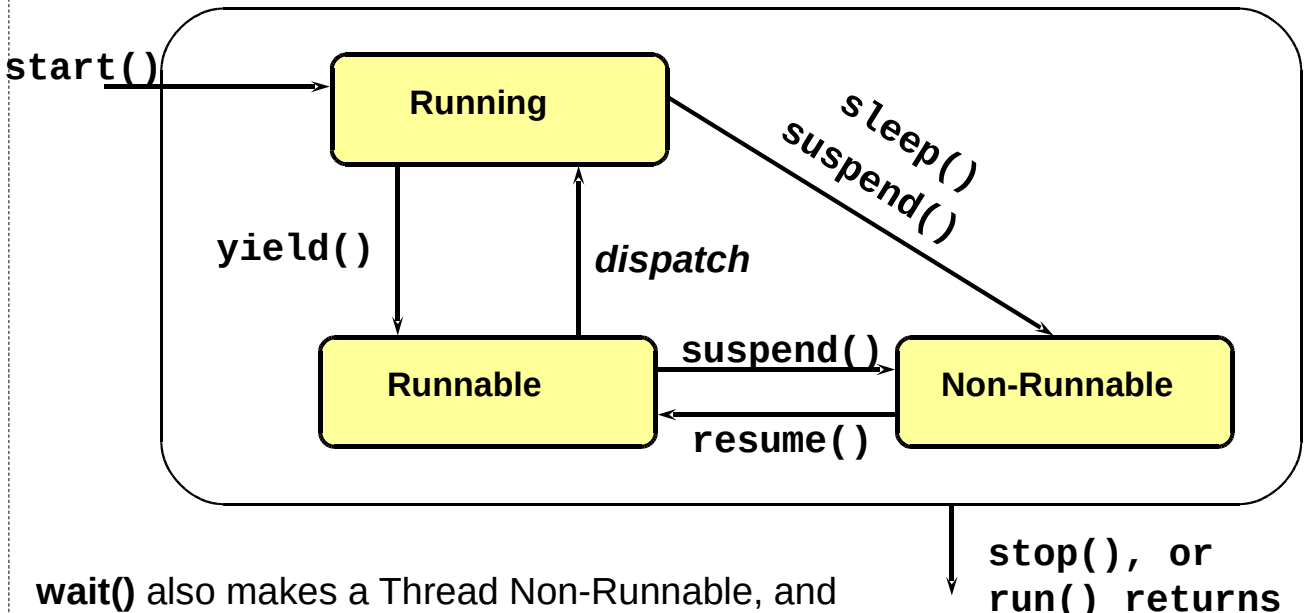


*Diagram from Magee/Kramer*



# States of a thread - 2

- Once started an alive thread has a number of substates




**wait()** also makes a Thread Non-Runnable, and **notify()** Runnable. Both these methods are from the **Object** class which is the parent class of every class in Java. wait() and notify() will be covered in depth in the later lectures



*Diagram from Magee/Kramer*

# Java Thread API

- The Thread class has many other methods apart from *start()* and run*()*
- Following is the list of commonly used methods:
  - *sleep(time in ms)* – make the thread sleep for some time
  - *getId()* and getName() - get ID and name of thread
  - *getPriority()* - priority of thread
  - *getState()* - current state of the thread
  - *isAlive()* - check if a thread is alive
  - *yield()* - makes the runnable thread to give the CPU to another thread
  - *join()* - makes the caller wait for another thread to complete – this is very useful when breaking a large problem into smaller tasks and collecting the results from them



# Resources and references

- Concurrency: State Models & Java Programs, 2e, Jeff Magee & Jeff Kramer, Wiley
- Java Thread API - *http://bit.ly/jthreadapi*

  Look at the Thread API page to understand which methods to use and know what methods have been deprecated
- Java Concurrency tutorial – is the official Java tutorial for using the Thread API and concurrency in Java - *http://bit.ly/jconctut*
- *Course Website – http://sfdv3006.wikidot.com*