THE GEORGE WASHINGTON UNIVERSITY

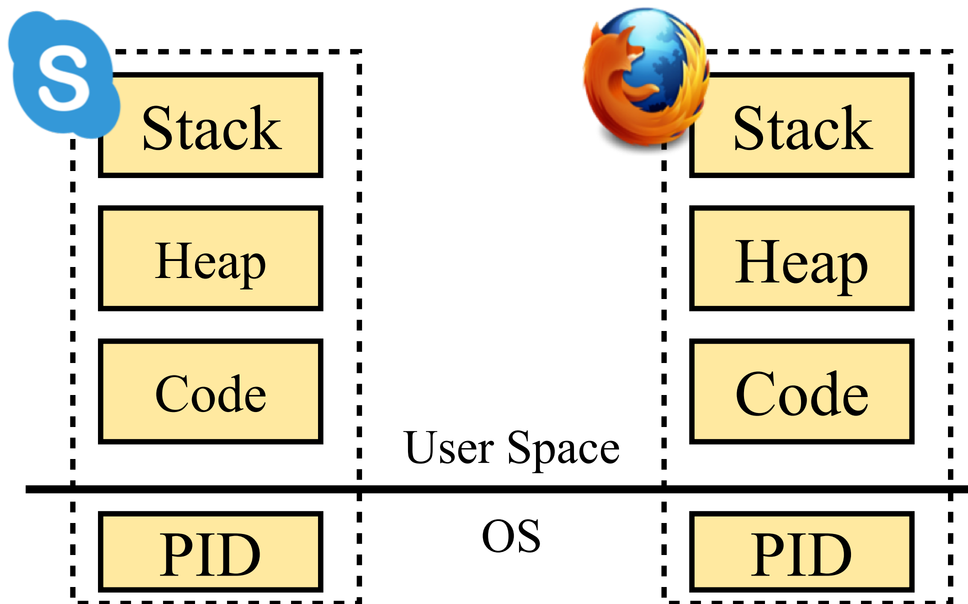
WASHINGTON, DC

Multithreading

Clone https://github.com/cs2113f18/template-j-7

Threads and Processes

- Operating system schedules **processes**.
- Each process has own resources and code.
- Switching the active process is (relatively) slow.



Task Manager

File Options View

Processes | Performance | App history | Startup | Users | Details | Services

Name	22% CPU	31% Memory	0% Disk	0% Network
Apps (6)				
HeavyLoad	0%	7.6 MB	0 MB/s	0 Mbps
Microsoft Edge	0%	10.8 MB	0 MB/s	0 Mbps
PicPick (32 bit)	0%	56.7 MB	0 MB/s	0 Mbps
Task Manager	0.5%	29.7 MB	0 MB/s	0 Mbps
Windows Command Processor	0%	0.4 MB	0 MB/s	0 Mbps
Windows Explorer (5)	0.3%	58.5 MB	0 MB/s	0 Mbps
Between PCs				
Between PCs				
HomeGroup				
picpick_portable				
View and print your homegroup password				
Background processes (44)				
Application Frame Host	0%	12.7 MB	0 MB/s	0 Mbps
Browser_Broker	0%	2.4 MB	0 MB/s	0 Mbps

Fewer details Restart

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Threads and Processes

- Threads allow **concurrency** within one app.
- Fast to switch between.
- Threads share the same memory space.

Heap

Code

T1

T2

Thread 1

Thread 2

Stack

Stack

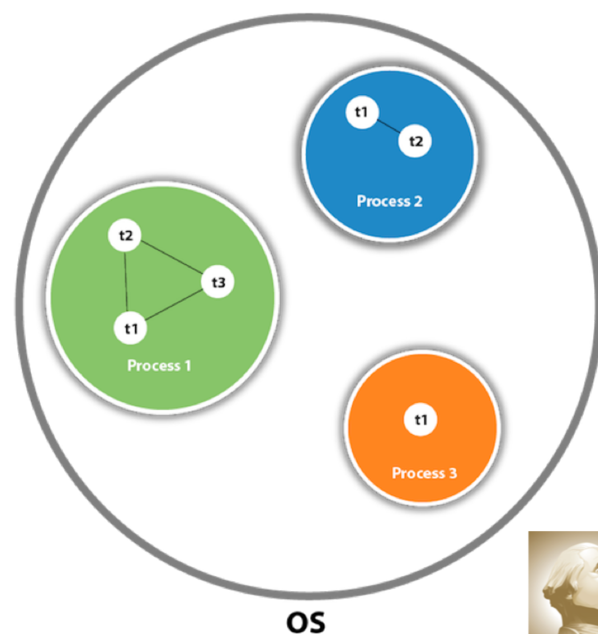
PID

Thread and Processes

- A thread is a lightweight sub-process, the smallest unit of processing.
- A thread is a separate path of execution.
- Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.

As shown in the right figure, a thread is executed inside the process.

There can be multiple processes inside the OS, and one process can have multiple threads.



Thread in Java

When an application first begins, user (main) thread is created.

```
import java.util.ArrayList;

public class Books {

    public static void main(String[] args) {

        ArrayList<String> listOfBooks = new ArrayList<>();
        listOfBooks.add("Head First Java");
        listOfBooks.add("A Game of Thrones");
        listOfBooks.add("Effective Java");
        listOfBooks.add("The Shining");
        listOfBooks.add("Silent Hill");

        System.out.println("List before : " + listOfBooks);

        //Using forEach loop to iterate and removing
        //element during iteration will throw
        //ConcurrentModificationException in Java
        for(String book: listOfBooks) {
            if(book.contains("Java")) {
                listOfBooks.remove(book);
                System.out.println("Removing " + book);
            }
        }

        System.out.println("List After : " + listOfBooks);

    }
}
```

Advantages of single thread:

- Reduces overhead in the application as single thread execute in the system.
- Also, it reduces the maintenance cost of the application.

```
Exception in thread "main" List before : [Head First Java
Removing Head First Java
java.util.ConcurrentModificationException
    at java.base/java.util.ArrayList$Itr.checkForComodifi
    at java.base/java.util.ArrayList$Itr.next(ArrayList.j
    at Books.main(Books.java:19)
```

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Multithreading in Java

- It is a process of executing two or more threads simultaneously.
- It is also known as Concurrency in Java.
- Each thread runs parallel to each other.
- The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time.

java.lang

Class Thread

java.lang.Object
    java.lang.Thread

All Implemented Interfaces:

Runnable

Direct Known Subclasses:

ForkJoinWorkerThread

Method	Description
start()	This method starts the execution of the thread and JVM calls the run() method on the thread.
Sleep(int milliseconds)	This method makes the thread sleep hence the thread's execution will pause for milliseconds provided and after that, again the thread starts executing. This help in synchronization of the threads.
getName()	It returns the name of the thread.
setPriority(int newpriority)	It changes the priority of the thread.
yield ()	It causes current thread on halt and other threads to execute.

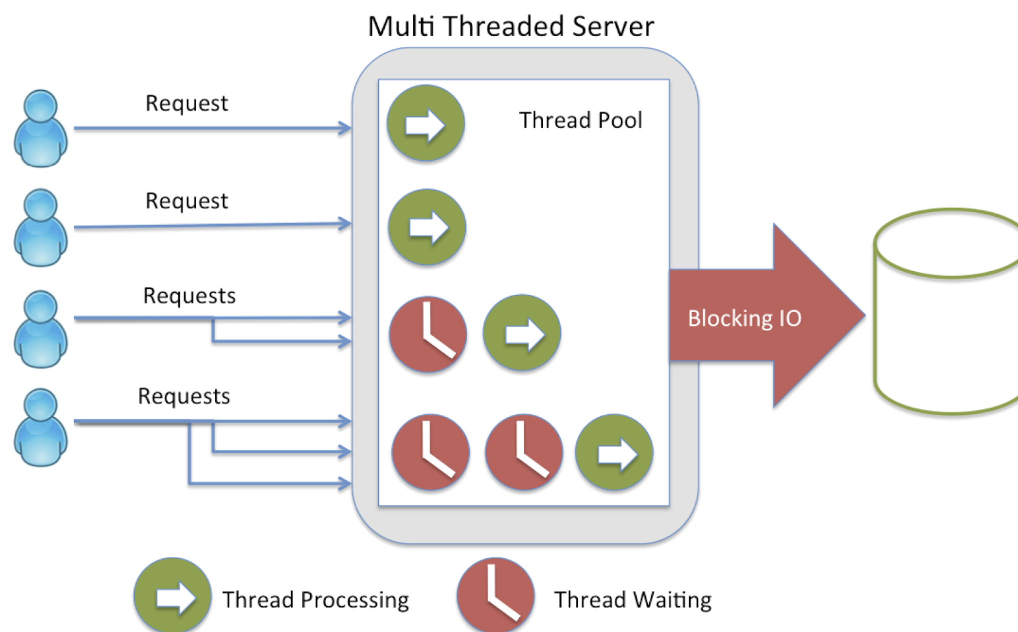
```
public class Thread
extends Object
implements Runnable
```

A *thread* is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

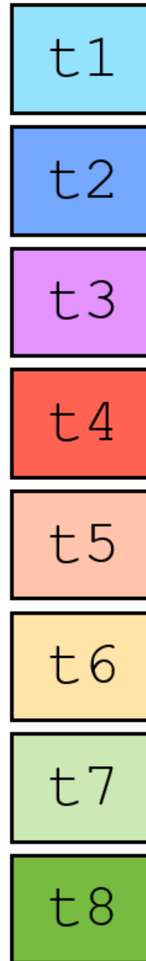
Why Using Multithreading

- To make a task run parallel to another task. e.g. drawing and event handling.
- To take full advantage of CPU power.
- For reducing response time.
- To sever multiple clients at the same time.



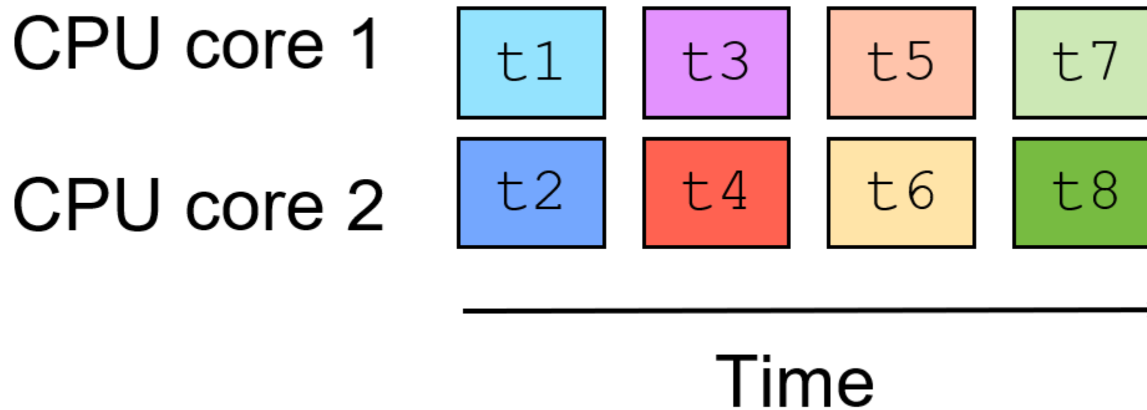
Multithreading in Java

- If we start 8 threads, will they all **run at once?**



Multithreading in Java

- If we start 8 threads, will they all **run at once?**
  - **No!**



- Only run one thread per CPU core at a time

Threads Have Their Own Stacks

1) When we enter main() method

main

stack 1 (main thread's)

As soon as main is called by JVM it is pushed on Stack

2) When main() calls method1() method

method1

main

stack 1 (main thread's)

As soon as main calls method1 (), method1 pushed on Stack

3) When methd1() calls thread.start()

method

main

stack 1
(main thread's)

run()

stack 2
(Thread-1's)

method1() creates new thread by calling thread.start (), as threads have their own stack - new stack is created.

```
class MyThread extends Thread{
    public void run(){
        System.out.println("in run() method");
    }
}

public class MyClass {
    public static void main(String...args){
        System.out.println("In main() method");
        method1();
    }

    static void method1(){
        MyThread obj=new MyThread();
        obj.start();
    }
}

/*OUTPUT

currentThreadName= main
in run() method
currentThreadName= Thread-1

*/
```

https://www.javamadesoeasy.com/2015/03/threads-implement-their-own-stack.html

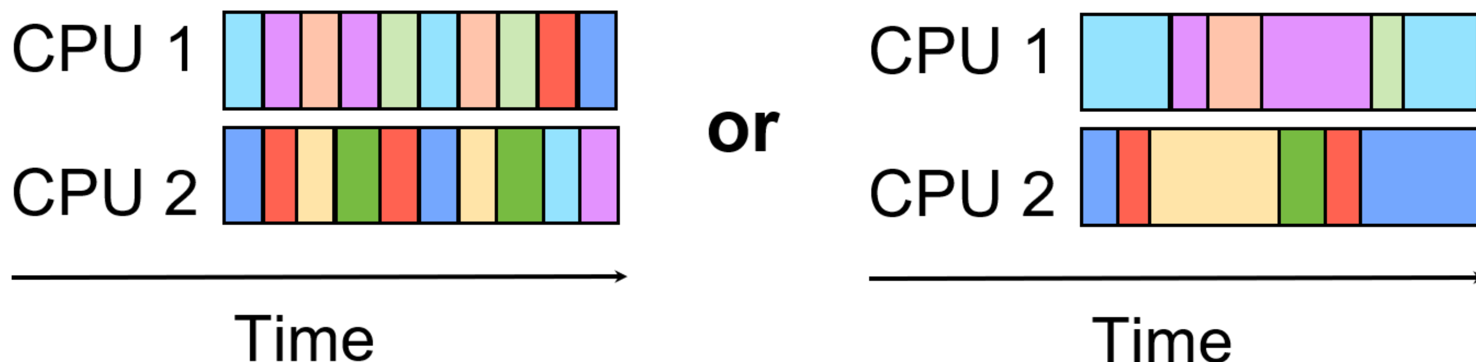
Who decides the order?

- Take a look at **threads.scheduling.ManyThreads.java**
- What happens when you run the code?
- Do you get the same ordering as your neighbor?
- What if you change the number of iterations?
- Or the number of threads?
- Is it the same every time you repeat?

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Scheduling Threads

- Ordering may be very random



- Threads can explicitly relinquish the CPU
- **Scheduler** can interrupt and pick someone else
- Do not assume a particular ordering!!

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Creating a Thread in Java

There are two ways to create a new thread of execution:

- Declare a class to be a subclass of Thread. This subclass should override the run method of class Thread.

```
public class MyThread extends Thread {
   public void run(){
    System.out.println("thread is running...");
  }
   public static void main(String[] args) {
     MyThread obj = new MyThread();
     obj.start();
}
```

- The class should extend Java Thread class.
- The class should override the run() method.
- The functionality that is expected by the Thread to be executed is written in the run() method.

- Declare a class that implements the Runnable interface. That class then implements the run method.

```
public class MyThread implements Runnable {
   public void run(){
     System.out.println("thread is running..");
   }
   public static void main(String[] args) {
     Thread t = new Thread(new MyThread());
     t.start();
}
```

- The class should implement the Runnable interface.
- The class should implement the run() method in the Runnable interface.
- The functionality that is expected by the Thread to be executed is put in the run() method.

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Extends Thread Class vs Implements Runnable Interface

- Extending the Thread class will make your class unable to extend other classes, because of the single inheritance feature in JAVA. However, this will give you a simpler code structure.

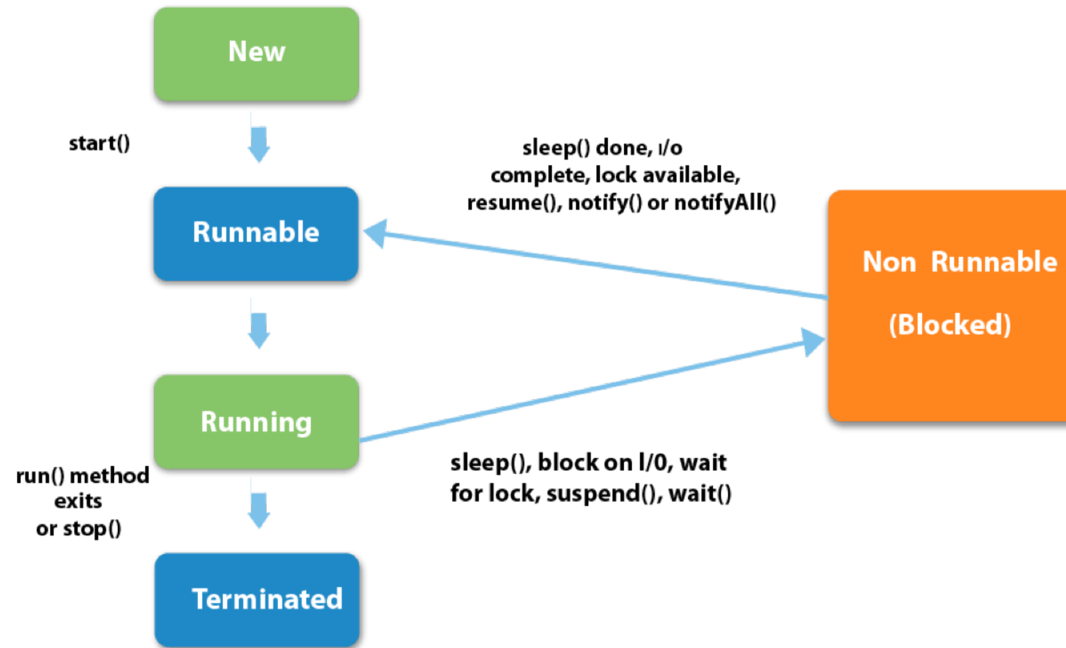
- If you implement Runnable, you can gain better object-oriented design and consistency and also avoid the single inheritance problems. (preferred)

```java
// Java program to illustrate defining Thread
// by implements Runnable interface
class Geeks {
    public static void m1()
    {
        System.out.println("Hello Visitors");
    }
}

// Here we can extends any other class
class Test extends Geeks implements Runnable {
    public void run()
    {
        System.out.println("Run method executed by child Thread");
    }
    public static void main(String[] args)
    {
        Test t = new Test();
        t.m1();
        Thread t1 = new Thread(t);
        t1.start();
        System.out.println("Main method executed by main thread");
    }
}
```

https://www.geeksforgeeks.org/implement-runnable-vs-extend-thread-in-java/

Thread Life Cycle in Java



- A NEW Thread (or a Born Thread) is a thread that's been created but not yet started. It remains in this state until we start it using the start() method.
- When we've created a new thread and called the start() method on that, it's moved from NEW to RUNNABLE state. Threads in this state are either running or ready to run, but they're waiting for resource allocation from the system.
- A thread is in the BLOCKED state when it's currently not eligible to run. It enters this state when it is waiting for a monitor lock, waiting for some other thread to perform a particular action, and is trying to access a section of code that is locked by some other thread.
- It's in the TERMINATED (DEAD) state when it has either finished execution or was terminated abnormally.

https://www.javatpoint.com/life-cycle-of-a-thread

Why don't we call run() method directly？

Take a look at

threads.examples.RunMethodExamples.java
threads.examples.RunMethodExamples2.java

Calling run() method:

```
public class RunMethodExample implements Runnable{
   public void run(){
      for(int i=1;i<=3;i++){
        try{
              Thread.sleep(1000);
          }catch(InterruptedException ie){
                ie.printStackTrace();
           }
         System.out.println(i);
      }
   }
   public static void main(String args[]){
      Thread th1 = new Thread(new RunMethodExample(), "th1");
      Thread th2 = new Thread(new RunMethodExample(), "th2");
      th1.run();
      th2.run();
   }
}
```

Calling strat() method:

```
public class RunMethodExample2 {
   public void run(){
      for(int i=1;i<=3;i++){
         try{
               Thread.sleep(1000);
         }
         catch(InterruptedException ie){
               ie.printStackTrace();
         }
         System.out.println(i);
      }
   }
   public static void main(String args[]){
      Thread th1 = new Thread(new RunMethodExample(), "th1");
      Thread th2 = new Thread(new RunMethodExample(), "th2");
      th1.start();
      th2.start();
   }
}
```

https://beginnersbook.com/2015/03/why-dont-we-call-run-method-directly-why-call-start-method/

Can we start a Thread twice in Java?

The answer is **no**, once a thread is started, it can never be started again. Doing so will throw an IllegalThreadStateException. Lets have a look at the below code:

Take a look at
threads.examples.ThreadTwiceExample.java

```
public class ThreadTwiceExample implements Runnable {
   @Override
   public void run(){
        Thread t = Thread.currentThread();
        System.out.println(t.getName()+" is executing.");

   }
   public static void main(String args[]){
        Thread th1 = new Thread(new ThreadTwiceExample(), "th1");
        th1.start();
        th1.start();
   }
}
```

https://beginnersbook.com/2015/03/can-we-start-a-thread-twice-in-java/

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Networking Threads

- Server Pseudocode:

```
Create a ServerSocket
while (true):
        Wait for client to connect
        Setup new socket for client
        Read request from client
        Prepare response
        Send response to client
        close socket for client
```

```
ServerSocket server = new
  ServerSoket(portnum);
```

```
Socket sock =
serverSocket.accept();
```

```
// Use BufferedReader
and PrintWriter classes
```

- What happens if preparing a response is very slow? What if there is more than one client?

Networking Threads

- Server Pseudocode:

```
Create a ServerSocket
while (true):
    Wait for client to connect
    Setup new socket for client
    start new thread to do -->
```

```
Read request from client
Prepare response
Send response to client
close socket for client
```

- What happens if preparing a response is very slow? What if there is more than one client?

THE GEORGE WASHINGTON UNIVERSITY
WASHINGTON, DC

Networking Threads

- Server Pseudocode:

```
Create a ServerSocket
while (true):
   Wait for client to connect
   Setup new socket for client
   start new thread to do -->
```

```
Read request from client
Prepare response
Send response to client
close socket for client
```

- What happens if prepa… slow? What if there is …

Networking Threads

- Main thread
  - Creates server sockets and waits for clients to connect
  - Start a new worker thread to process each client
- Worker thread
  - Needs a new class that implements Runnable
  - run() method processes a single request from the client

```
Create a ServerSocket
while (true):
   Wait for client to connect
   Setup new socket for client
   Read request from client
   Prepare response
   Send response to client
   close socket for client
```

Do these in a thread!

Threaded Web Server

- Work with a neighbor
- Look at the **threads.web.SlowWebServer** class
  - How does it work?
  - Why is it slow?
  - Have one person run it and then both connect at the same time with a web browser
  - Use the **FrameBrowser** class to load the page and measure the time it takes. Does the time depend on whether another request is active?
- Make the Web Server use threads
  - Create and start a new thread for every new user

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC

Chat Threads

- Suppose you want to build a chat server

```
Server

wait for client
while true:
read message
print message
send message back
```

Remember: whenever you call readLine the current thread will wait, preventing it from doing anything else. That means if you need to call readLine to get data from N different clients, you want to do that in N different threads!

```
Client 1
Connect: connect to server
Send: send message
Another thread:
while(true)
read from BR
```

```
Client 2
Connect: connect to server
Send: send message
Another thread:
while(true)
read from BR
```

Summary

- Threads let us split a program into multiple tasks that are run (seemingly) simultaneously
  - The JRE scheduler determines the order threads are run in
  - If computer has multiple CPU cores, threads may run in parallel
- Threads are important for networking:
  - Server typically needs one thread per client
  - Client might need threads if it must multi-task
    - Remember, calling readLine() is blocking! If you aren't sure if there is data ready to read, you probably want to do this in a special thread!

THE GEORGE WASHINGTON UNIVERSITY

WASHINGTON, DC