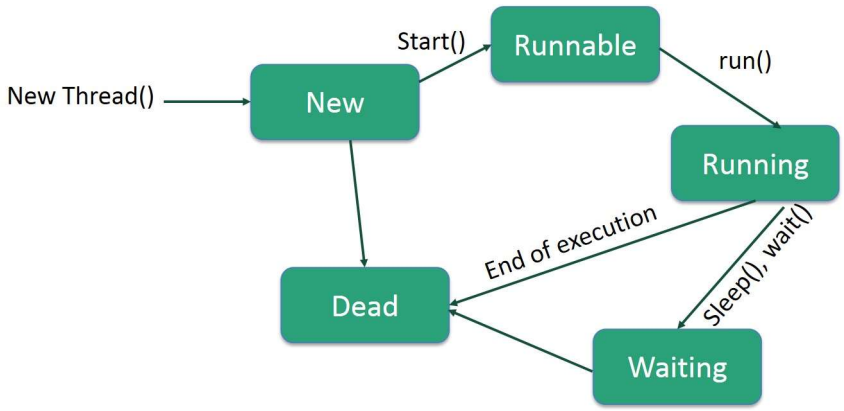
Multi-Threaded Applications with Java

CSCI 2300

Multi-Threading

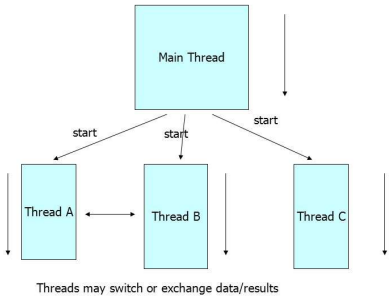
- Two ore more parts executing concurrently
- Each task handling a different part
- Shared resources:
  - Memory
  - CPU
  - Possibly objects/data
  - Synchronization may be needed

Life-Cycle of a Thread



Which of the threads in the diagram below terminates first?

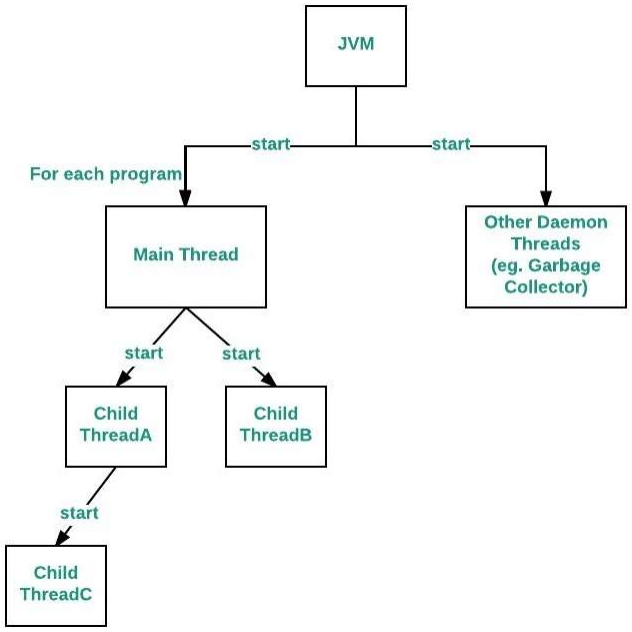
A Multithreaded Program



A. Thread A
B. Thread B
C. Thread C
D. Main Thread
E. Cannot be determined with the information provided

Threads in Java

- Main thread – starts when 'main' method is executed
- Garbage Collector
  - "behind the scenes" thread
  - Reclaims allocated memory that is no longer used
  - Don't need to call 'delete' or 'free' as in C++



Implementation in Java using Runnable

- Implement a `Runnable` interface

void	**run**() When an object implementing interface Runnable is used to create a thread, starting the thread causes the object's run method to be called in that separately executing thread.

- Instantiate a Thread object using one of Thread's constructors:

Thread(**Runnable** target)Allocates a new Thread object.
Thread(**Runnable** target, **String** name)Allocates a new Thread object.

- Call '`start()`' on the `Thread` object, which will call `run()` of the `Runnable` target

Example

```
public class Sleepy implements Runnable{
   protected Thread thread;
   …
   public void run(){
      //operations of this tread
   }
   public void start(){
      if (thread == null){
         thread = new Thread(this);
         thread.start();
      }
   }
}
```

Instantiating and Starting Sleepy

```
Sleepy sleepy = new Sleepy("Bob");
sleepy.start();
// other operations
```

- In a single threaded program, execution is blocked until the method terminates
- `sleepy.start()` will create a thread and call `sleepy.run()`
- Execution will proceed to "`// other operations`" without waiting for `sleepy.run()` to terminate
- Review `Runnable` example linked to today's lecture on the schedule

Code Example online: What is a possible outcome of running this code (before Enter is pressed)?

A. Bob: Sleeping
   Alice: Sleeping
   Bob: Awake
   Bob: Sleeping

B. Bob: Sleeping
   Bob: Awake
   Alice: Sleeping
   Bob: Sleeping
   Alice: Awake

C. Bob: Sleeping
   Alice: Sleeping
   Alice: Awake
   Alice: Sleeping
   Bob: Awake
   Bob: Sleeping

D. All of the above are possible

What will happen when Enter is pressed?

A. "Bob: stopped" will be printed before "Alice: stopped"
B. "Alice: stopped" will be printed before "Bob: stopped"
C. The behavior is not deterministic
D. It is possible to have "awake" and "sleeping" messages between "stopped" messages
E. A & D

Implementation in Java extending Thread

- Create a sub-class of the Java `Thread` class
- Override `run()` method of the `Thread` class
- Call `start()` method on an object of your new class.
  - This will call the `run()` method you overrode
- The new class inherits all public and protected methods of `Thread`
- The new class can be substituted for where `Thread` object is expected (Liskov Substitution Principle)
- Review `Thread` example linked to today's lecture on the schedule

Advantages and Disadvantages of the two approaches

A. Implementing `Runnable` gives us the flexibility to extend some other class (if needed)
B. Extending `Thread` allows us to use all public methods of `Thread` class on our new class
C. A & B
D. None of the above

Precautions: synchronization of shared resources is important

```
Balance: 100
Bob:   withdraw(100)
Alice: withdraw(100)
```

```
withdraw(d){
      movl balance , %eax
      subl d , %eax
      movl %eax , balance
}
```

Application is reading from a database. The process of reading from a database takes a long time. While reading from the database, your application could be doing some other operations. You are considering placing "read from the database" operation in a separate thread. Is this a good idea?

A. Yes, if "other operations" don't depend on the read from the database.
B. No, because reading from the database takes a long time.
C. No, because this is dangerous.

You are writing an application that has two objects: Player and Game. As soon as Player makes a move, the Game needs to update its state. After Game state is updated, the Player can make the next move. You are considering using threads to speed up processing. Is this a good idea?

A. Yes, because multi-threading can help improve performance.
B. No, because multi-threading should not be used if you need to guarantee timing of events.
C. No, because multi-threading requires synchronization on shared data.

You are writing an applicating for analyzing a large dataset. You know that you can break the dataset into independent subsets, analyze the subsets and then combine the results. You are considering using multi-threading for this. Is this a good idea?

A. Yes, analyzing multiple subsets in parallel will be more efficient than analyzing the entire dataset at once.
B. No, subsets are part of one dataset, so they are a shared resource between threads and will require much synchronization.
C. No, this approach adds complexity and does not provide an advantage

Threads in CSCI 2300

- Next: client-server applications
- Server will have a “listening thread” for accepting client requests
- Server will have a “sending thread” for sending out responses
- Client will have a “sending thread”
- Client will have a “listening thread”