Lab5

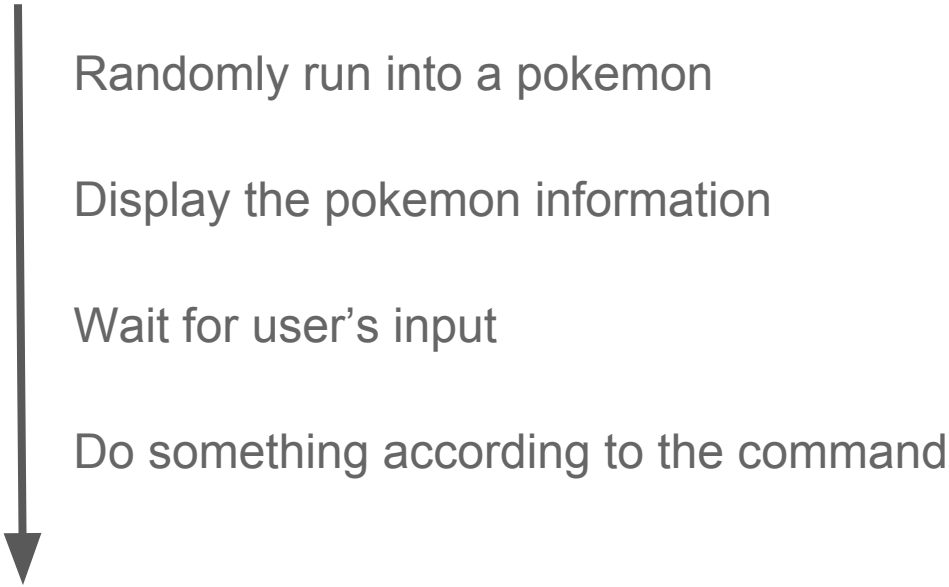
Multi-threaded Programming

Outline

- Single thread v.s multi-thread
- Life cycle
- Create a thread
- Daemon and non-daemon thread
- Multithreading Issues in Swing

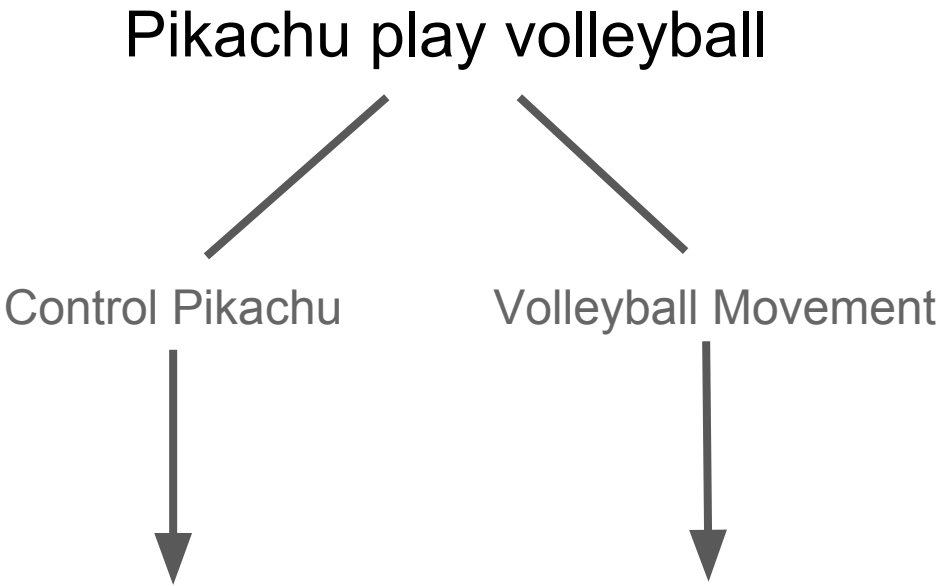
Single-Thread Process

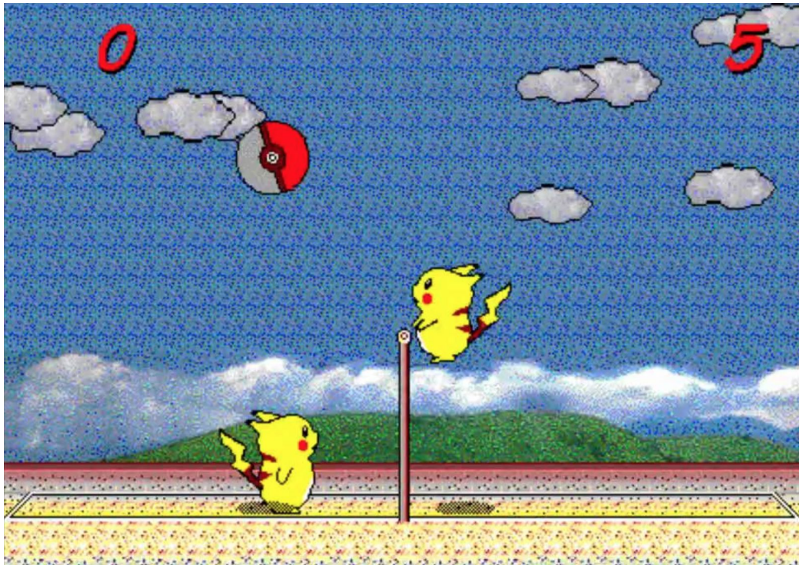
Catch the pokemon

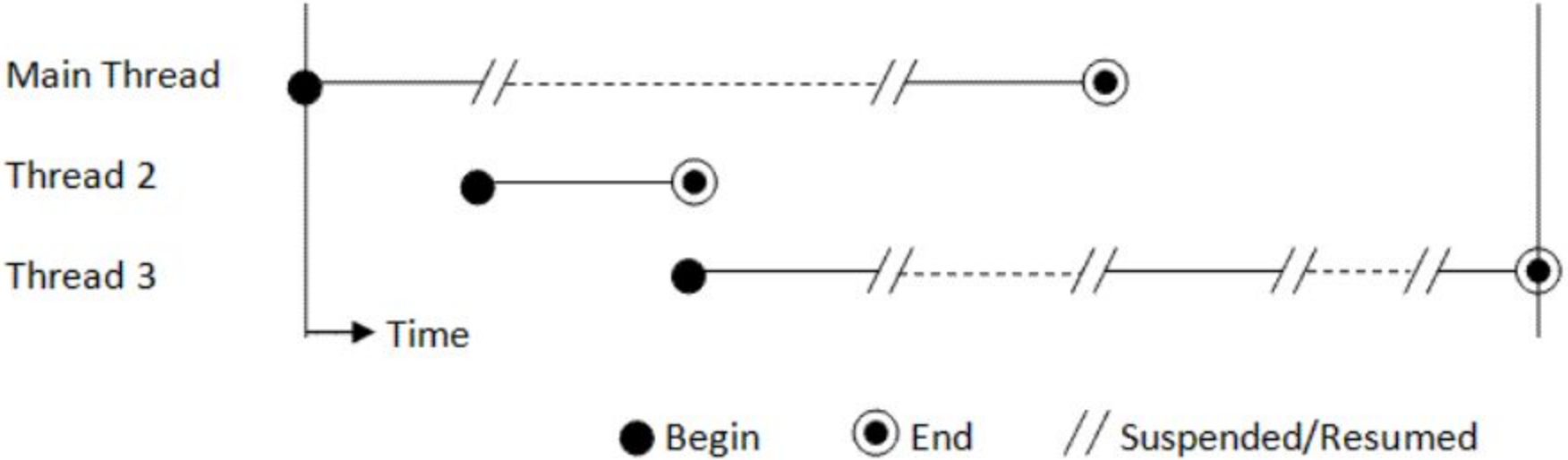


When a Java program starts up, one thread begins running immediately.

Multi-Thread Process



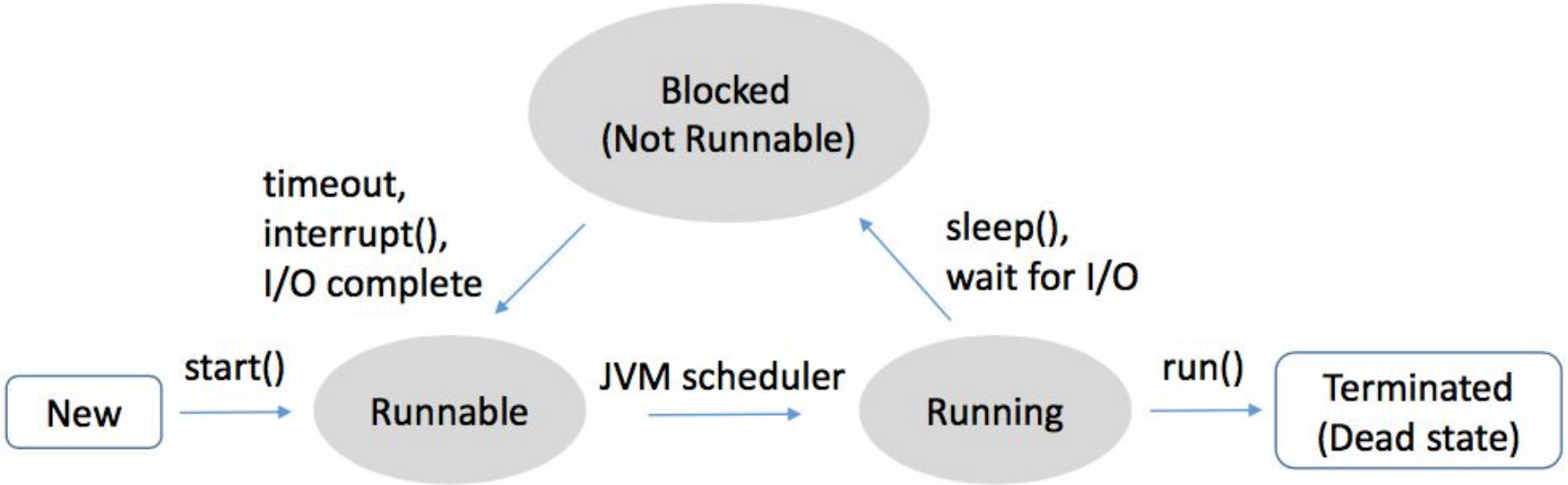




A program with 3 threads running under a single CPU.

https://www3.ntu.edu.sg/home/ehchua/programming/java/j5e_multithreading.html

Life Cycle


Blocked
(Not Runnable)
timeout,
interrupt(),
I/O complete
sleep(),
wait for I/O
New
start()
Runnable
JVM scheduler
Running
run()
Terminated
(Dead state)

Create a Thread

- Extend the *Thread* class
- Implement the *Runnable* interface

Extend *Thread* Class

- The extending class must override the *run( )* method, which specify the thread's operations and is the entry point for the new thread.
- Create an instance of this new class. This instance is called a Runnable object. (Thread class itself implements Runnable interface)
- It must also call start( ) to begin execution of the new thread.

https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

```java
public class MainThread {
    public static void main(String[] args) {
        new ChildThread().start();
        for (int i = 0; i < 5; i++){
            System.out.println("Main thread: " + i);
        }
        System.out.println("Main thread finished.");
    }
}

public class ChildThread extends Thread{
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("Child thread: " + i);
        }
        System.out.println("Child thread finished.");
    }
}
```

Output 1

```
Main thread: 0
Main thread: 1
Child thread: 0
Child thread: 1
Child thread: 2
Child thread: 3
Child thread: 4
Child thread finished.
Main thread: 2
Main thread: 3
Main thread: 4
Main thread finished.
```

Output 2

```
Child thread: 0
Child thread: 1
Child thread: 2
Child thread: 3
Child thread: 4
Main thread: 0
Child thread finished.
Main thread: 1
Main thread: 2
Main thread: 3
Main thread: 4
Main thread finished.
```

Thread is executed randomly, so the output might not be the same.

Implement *Runnable* Interface

- In the class, provide implementation to the abstract method run() to specify the thread's operations.
- A client class creates an instance of this new class. The instance is called a Runnable object.
- Construct a new Thread object with the Runnable object as argument to the constructor.
- The start() called back the run() in the Runnable object.

https://docs.oracle.com/javase/7/docs/api/java/lang/Runnable.html

```java
public class MainThread {
    public static void main(String[] args) {
        Thread childThread = new Thread(new ChildThread());
        childThread.start();
        for (int i = 0; i < 5; i++){
            System.out.println("Main thread: " + i);
        }
        System.out.println("Main thread finished.");
    }
}

public class ChildThread implements Runnable{
    @Override
    public void run() {
        for (int i = 0; i < 5; i++) {
            System.out.println("Child thread: " + i);
        }
        System.out.println("Child thread finished.");
    }
}
```

Runnable Object

```
public class MainThread {

    public static void main(String[] args) {
        System.out.println("Main thread starts.");
        Thread childThread = new Thread(new Runnable() {
            @Override
            public void run() {
                System.out.println("Child thread starts.");
                for (int i = 0; i < 5; i++){
                    System.out.println("Child thread: " + i);
                }
                System.out.println("Child thread finished.");
            }
        });
        childThread.start();
        for (int i = 0; i < 5; i++){
            System.out.println("Main thread: " + i);
        }
        System.out.println("Main thread finished.");
    }
}
```

Anonymous inner class

Non-daemon Thread

```
public class MainThread {
    public static void main(String[] args) {
        Thread childThread = new Thread(new ChildThread());
        childThread.start();
        for (int i = 0; i < 5; i++){
            System.out.println("Main thread: " + i);
        }
        System.out.println("Main thread finished.");
    }
}

public class ChildThread implements Runnable{
    @Override
    public void run() {
        System.out.println("A");
    }
}
```

Output

```
Main thread: 0
Main thread: 1
Main thread: 2
Main thread: 3
Main thread: 4
Main thread finished.
A
A
A
A
A
```

ChildeThread print “A” continously!

Daemon Thread

- A daemon thread is supposed to provide a general service in the **background** as long as the program is running.

```
public class MainThread {
    public static void main(String[] args) {
        Thread childThread = new Thread(new ChildThread());
        childThread.setDaemon(true);
        childThread.start();
        for (int i = 0; i < 5; i++){
            System.out.println("Main thread: " + i);
        }
        System.out.println("Main thread finished.");
    }
}
```

Thread Interrupt

- It is a mechanism whereby a thread that is sleeping can be made to prematurely stop sleeping.
- InterruptedException is thrown when another thread interrupts the thread.

```java
public class MainThread {
    public static void main(String[] args) {
        Thread childThread =  new Thread(new ChildThread());
        childThread.start();
        childThread.interrupt();
    }
}

public class ChildThread implements Runnable{
   @Override
   public void run() {
       for (int i = 0; i < 5; i++){
           System.out.println("Child thread: " + i);
           try {
               Thread.sleep(1000); // sleep for 1 sec
               System.out.println("I'm sleeping");
           } catch (InterruptedException e) {
               System.out.println("thread interrupted!");
           }
       }

       System.out.println("Child thread finished.");
   }
}
```

Output

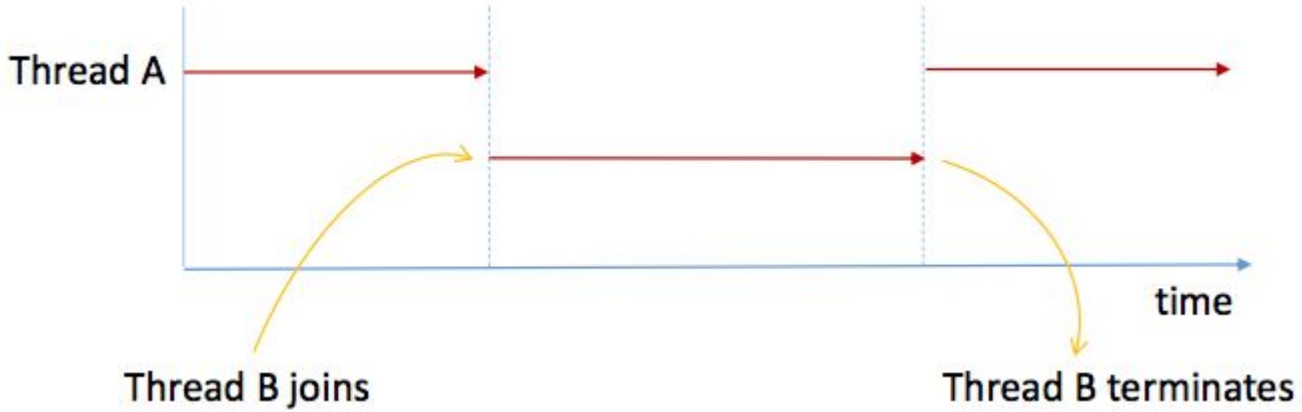
```
Child thread: 0
thread interrupt!
Child thread: 1
I'm sleeping
Child thread: 2
I'm sleeping
Child thread: 3
I'm sleeping
Child thread: 4
I'm sleeping
Child thread finished.
```

Thread Priority

- Range from 1(Thread.MIN_PRIORITY) to 10(Thread.MAX_PRIORITY)
- Default is 5(Thread.NORM_PRIORITY)
- Use setPriority(int priority) to set the priority of a thread.
- If the priority is not in the range between MIN_PRIORITY and MAX_PRIORITY, it will throw IllegalArgumentException.

Thread Join

- join(): Waits for the joined thread to terminate.
- join(long millis): Waits at most *millis* **millisecond** for the joined thread to terminate



```java
public class MainThread {
    public static void main(String[] args) {
        System.out.println("Main thread starts.");
        Thread childThread =  new Thread(new ChildThread());
        childThread.start();

        try {
            childThread.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Main thread finished.");
    }
}

public class ChildThread implements Runnable{
    @Override
    public void run() {
        System.out.println("Child thread starts.");
        for (int i = 0; i < 5; i++){
            System.out.println("Child thread: " + i);
        }
        System.out.println("Child thread finished.");
    }
}
```

Output

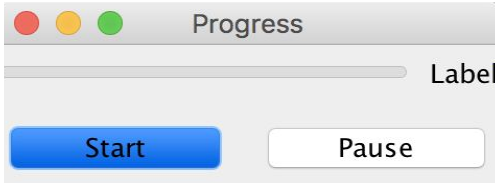
```
Main thread starts.
Child thread starts.
Child thread: 0
Child thread: 1
Child thread: 2
Child thread: 3
Child thread: 4
Child thread finished.
Main thread finished.
```

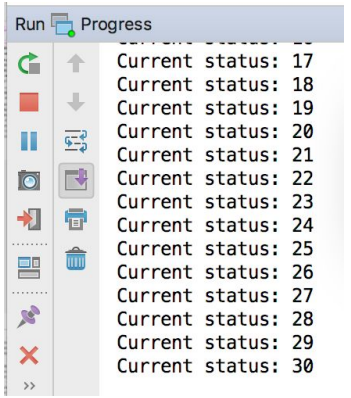
Multithreading Issues in Swing

- A Swing application runs on multiple threads.
  - Main Thread runs the main() method, starts the building GUI and exits.
  - Event-Dispatching Thread (EDT) handle the interaction with GUI elements.
  - Worker Thread is for compute-intensive task and I/O.

```
public Progress() {
    startBtn.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
            processing();
        }
    });
}

private void processing() {
    while (count < 100) {
        try {
            count++;
            Thread.sleep(100);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Current status: " + count);
    }
}
```

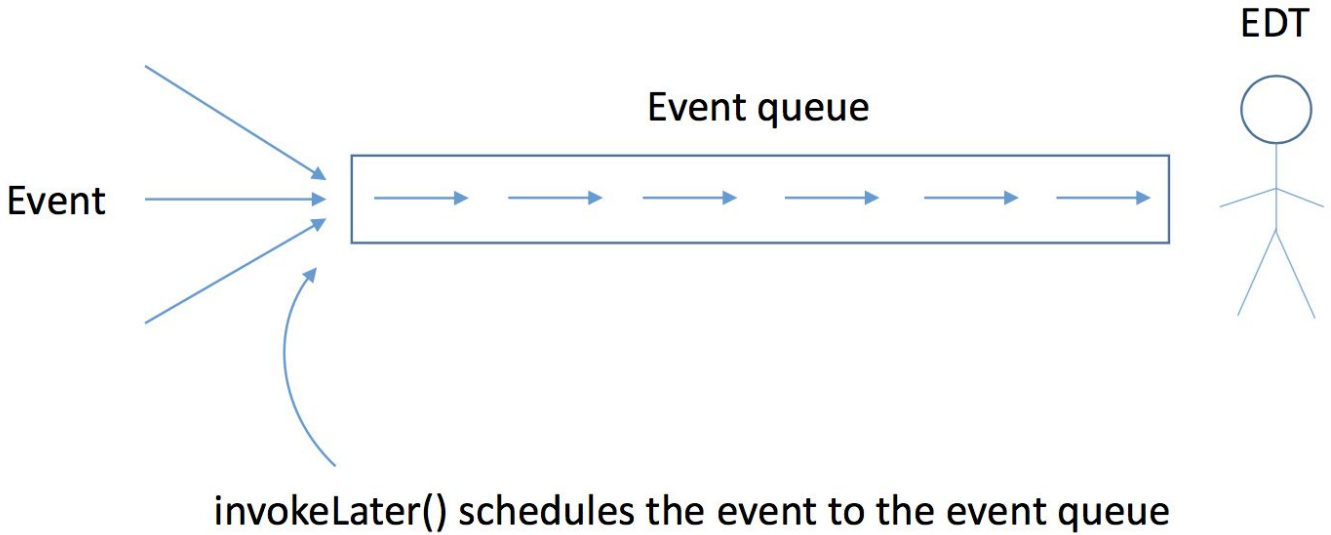




The screen freeze!

SwingUtilities.invokeLater()

- This method schedules the Runnable event in the event-dispatching thread.



```
    public Progress() {
        myThread = new Thread(new Processing());
        myThread.start();
    }

    class Processing implements Runnable {
        @Override
        public void run() {
            while (count < 100) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    System.out.println("pause");
                }
                if (isRunning) {
                    count++;
                    SwingUtilities.invokeLater(new Runnable() {
                        @Override
                        public void run() {
                            progressBar.setValue(count);
                            statusLabel.setText(String.valueOf(count));
                        }
                    });
                }
            }
        }
    }
}
```

SwingWorker Abstract Class

- How to Use Threads in Java Swing

invokeLater and SwingWorker

- Difference between SwingUtilities.invokeLater and SwingWorker<Void, Object>?
- Java - Difference between SwingWorker and SwingUtilities.invokeLater()
- Swing線程之SwingUtilities.invokeLater之解釋

Practice 5

Progress bar and Heavy computation simulator

Simulator: Pretend to be busy

- start -> disabled
- pause -> resuem ->pause

Use SwingUtilities.invokeLater to update GUI

https://github.com/alliechang/Practice5