Introduction to Java Programming Language

Junji Zhi

University of Toronto

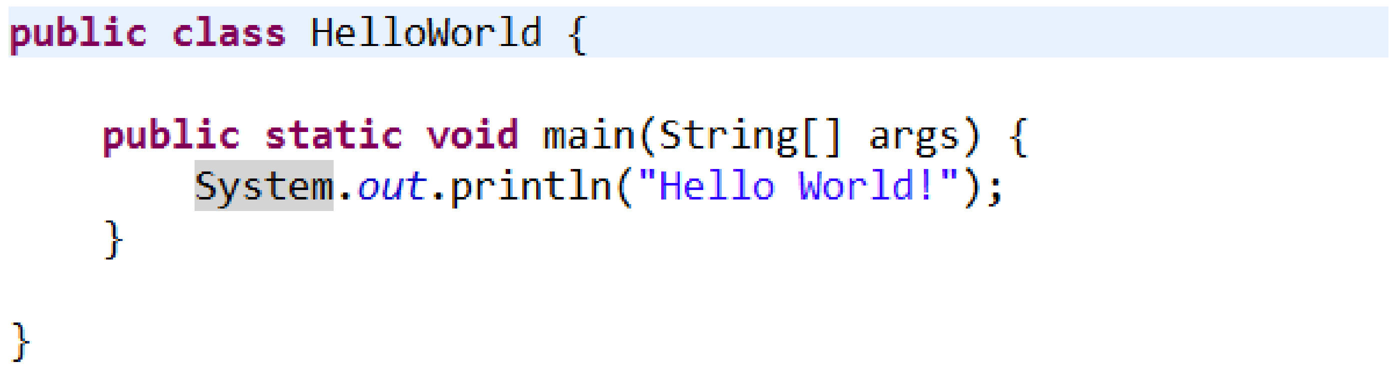
Content

- Java language Syntax
- “Hello World” program example
- Compiling, Running and Debugging Java code
- Inheritance
- Threading
- Synchronization

Java programming Language

- Some buzzwords for Java
  - “Write Once, Run Anywhere”
  - Simple
  - Object oriented
  - Multithreaded
  - Portable
  - High performance

Example: Hello World Program

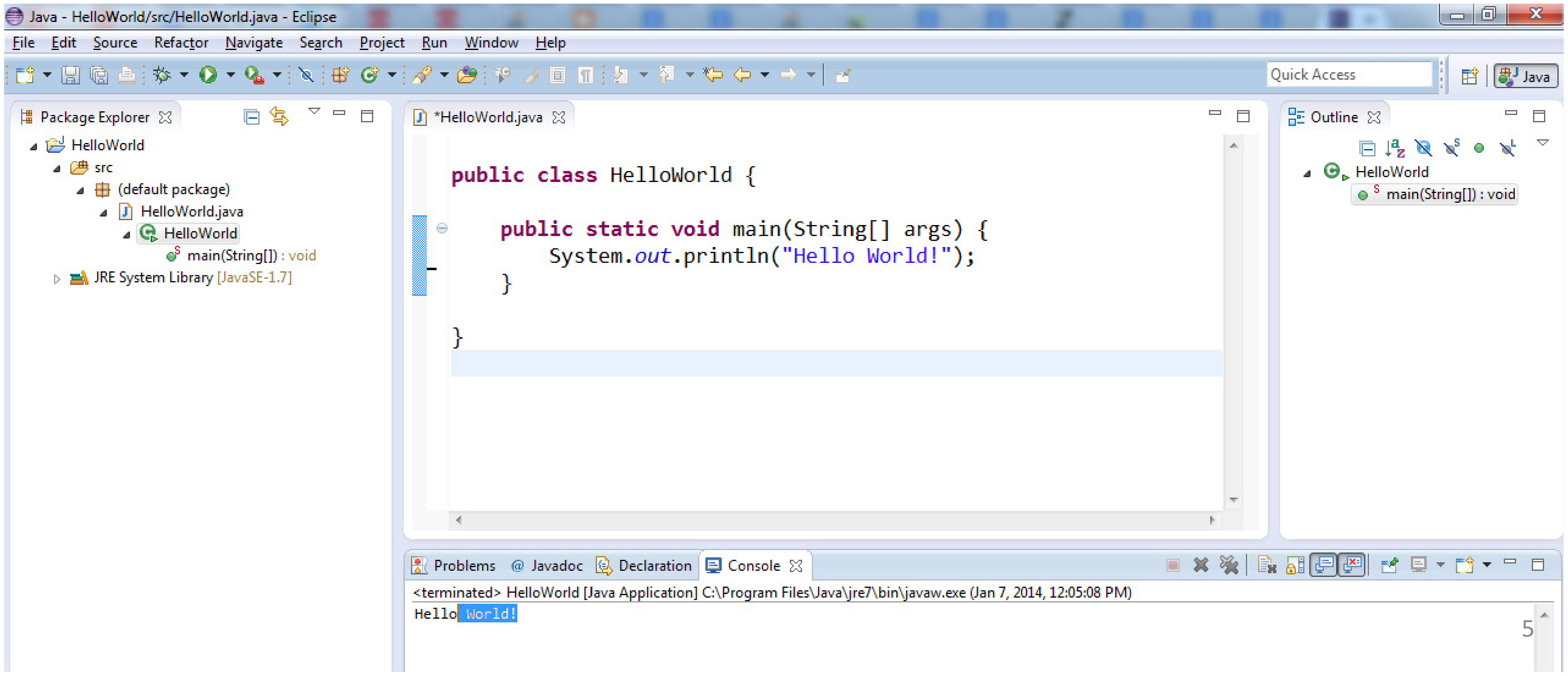


```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

- Everything is in a class
- In the runnable public class:
  - `public static void main(String [] args)`

Running HelloWorld

Eclipse Download from here.
IntelliJ IDEA Download from here.



```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }

}
```

```
<terminated> HelloWorld [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Jan 7, 2014, 12:05:08 PM)
Hello World!
```

Primitive Data Types

- **Primitive Data Types**:
  byte, short, int, long, float, double, boolean, char
- **Arrays** are also a class

```
long [] a = new  long[5];
```

  – You can get the length by visiting the length field of array object a, like this: **`a.length`**
- **String** class is very commonly used to represents character strings, for example

```
String s1 = “Hello ”, s2 = “Wolrd!”;
String s3 = s1 + s2;
```

Operators (same as C/C++) [3]

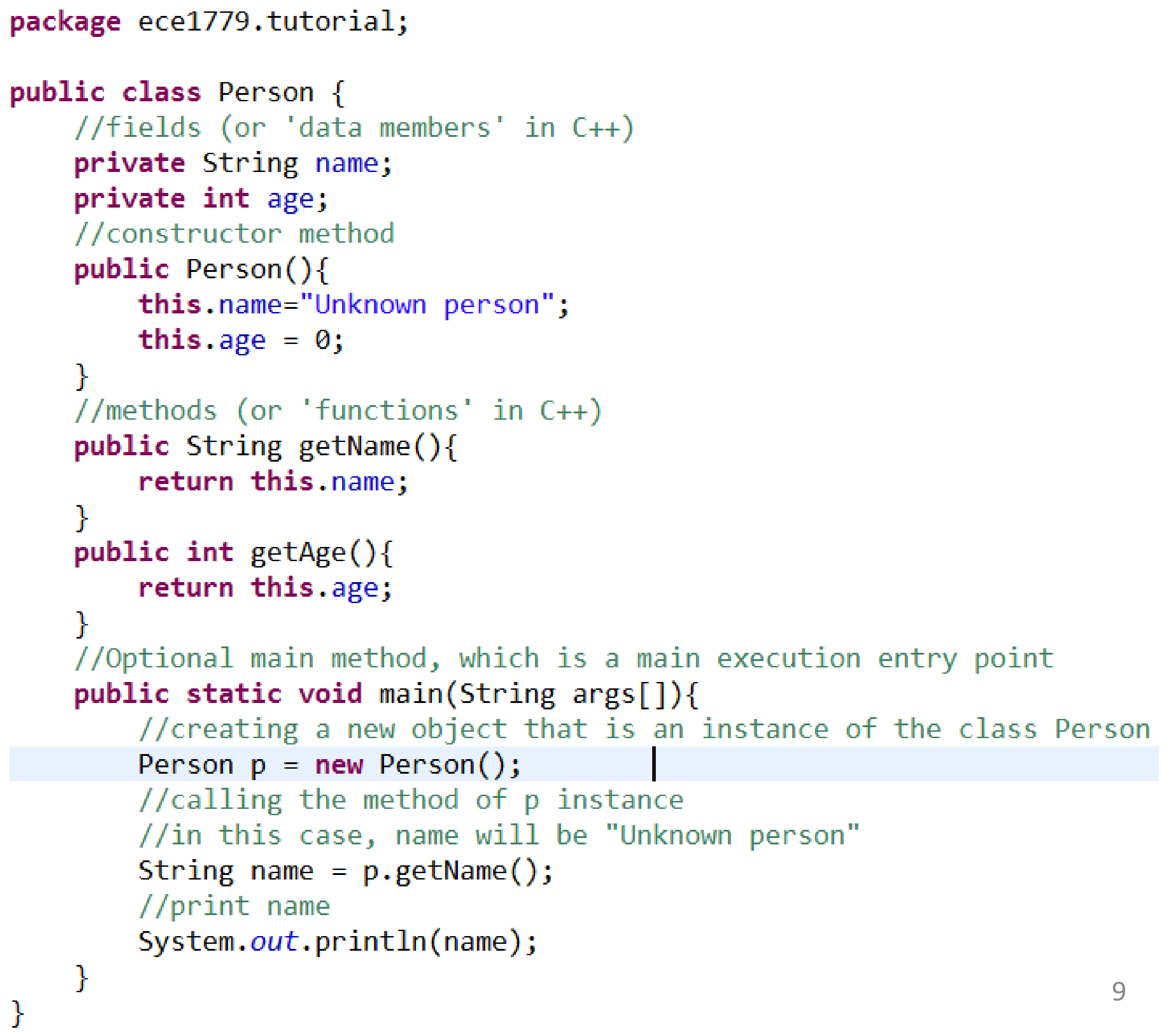
- ++,-- Auto increment/decrement
- +,- Unary plus/minus
- *,/ Multiplication/division
- % Modulus
- +,- Addition/subtraction

Declaring Variables [3]

```
int n = 1;
char ch = ‘A’;
String s = “Hello”;
Long L = new Long(100000);
boolean done = false;
final double pi = 3.14159265358979323846;
Employee joe = new Employee();
char [] a = new char[3];
Vector v = new Vector();
```

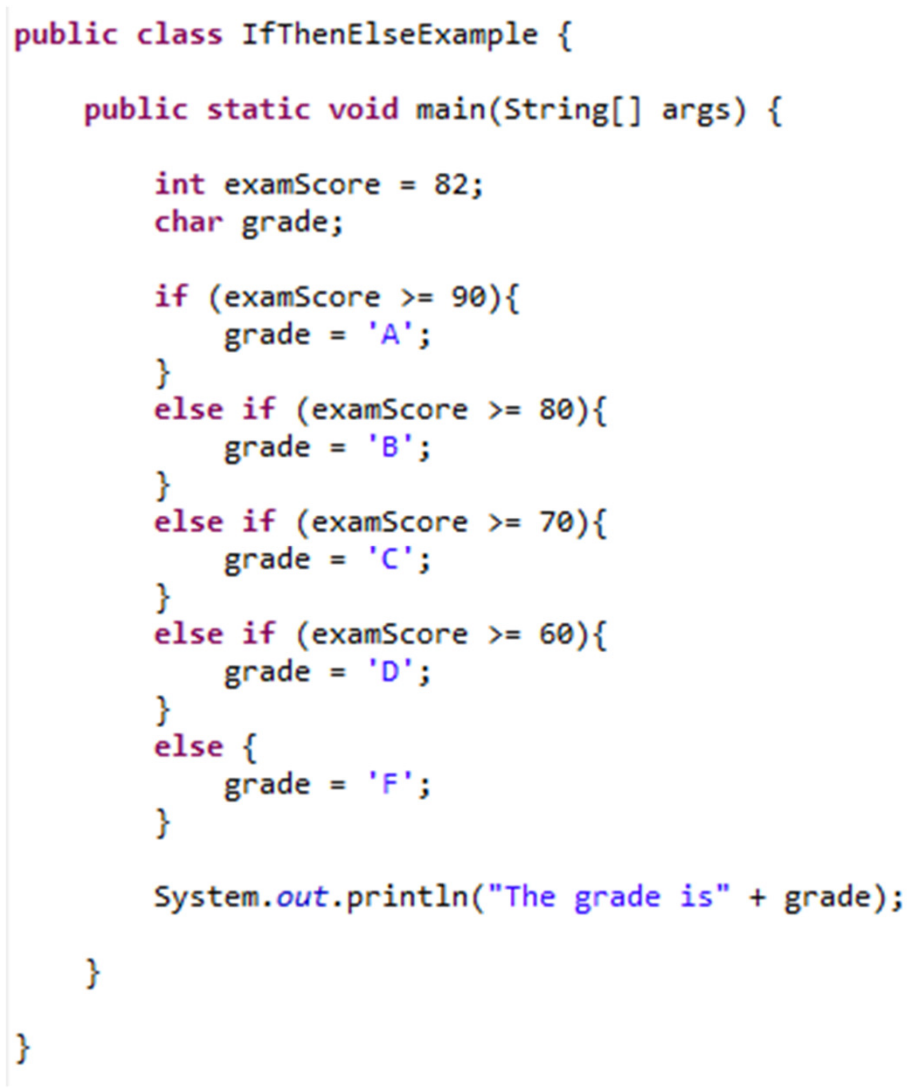
Declaring a class

- package
- Class name
- Constructor
- Fields
- methods



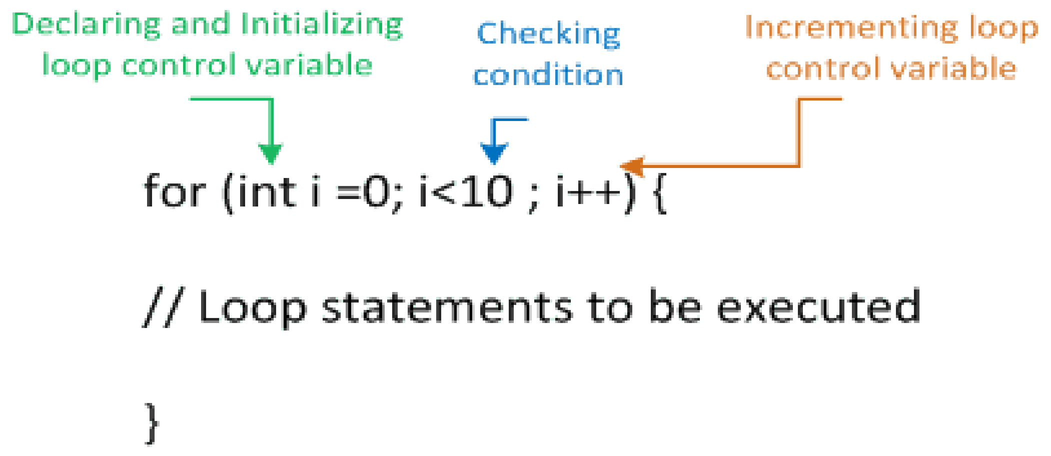
```
package ece1779.tutorial;

public class Person {
    //fields (or 'data members' in C++)
    private String name;
    private int age;
    //constructor method
    public Person(){
        this.name="Unknown person";
        this.age = 0;
    }
    //methods (or 'functions' in C++)
    public String getName(){
        return this.name;
    }
    public int getAge(){
        return this.age;
    }
    //Optional main method, which is a main execution entry point
    public static void main(String args[]){
        //creating a new object that is an instance of the class Person
        Person p = new Person();
        //calling the method of p instance
        //in this case, name will be "Unknown person"
        String name = p.getName();
        //print name
        System.out.println(name);
    }
}
```

Conditional Statements



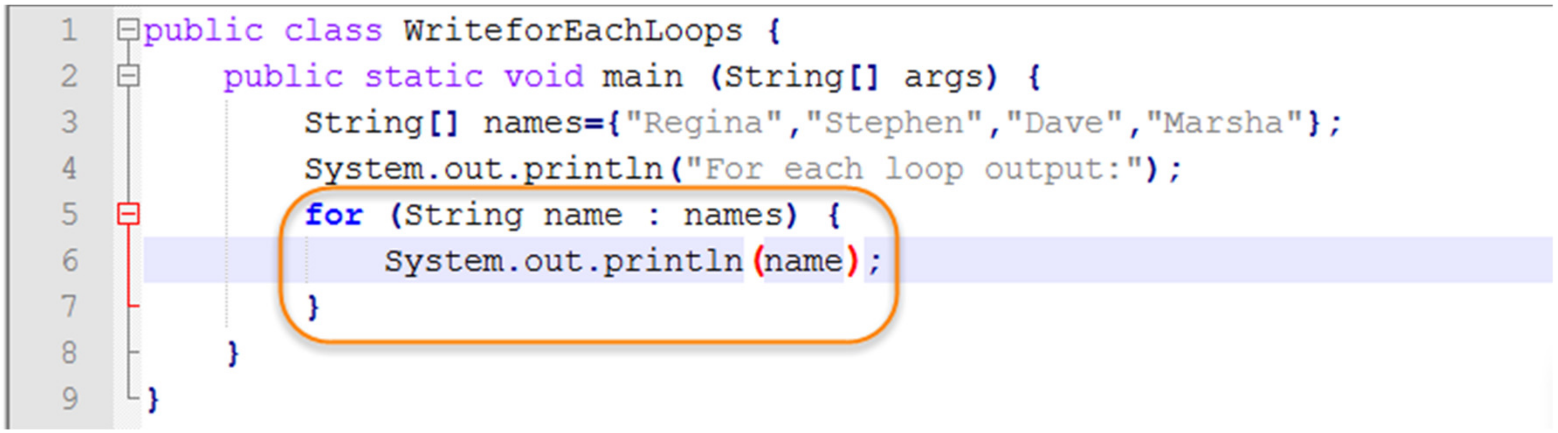
```java
public class IfThenElseExample {

    public static void main(String[] args) {

        int examScore = 82;
        char grade;

        if (examScore >= 90){
            grade = 'A';
        }
        else if (examScore >= 80){
            grade = 'B';
        }
        else if (examScore >= 70){
            grade = 'C';
        }
        else if (examScore >= 60){
            grade = 'D';
        }
        else {
            grade = 'F';
        }

        System.out.println("The grade is" + grade);

    }

}
```

Loop Statements


Declaring and Initializing loop control variable
Checking condition
Incrementing loop control variable

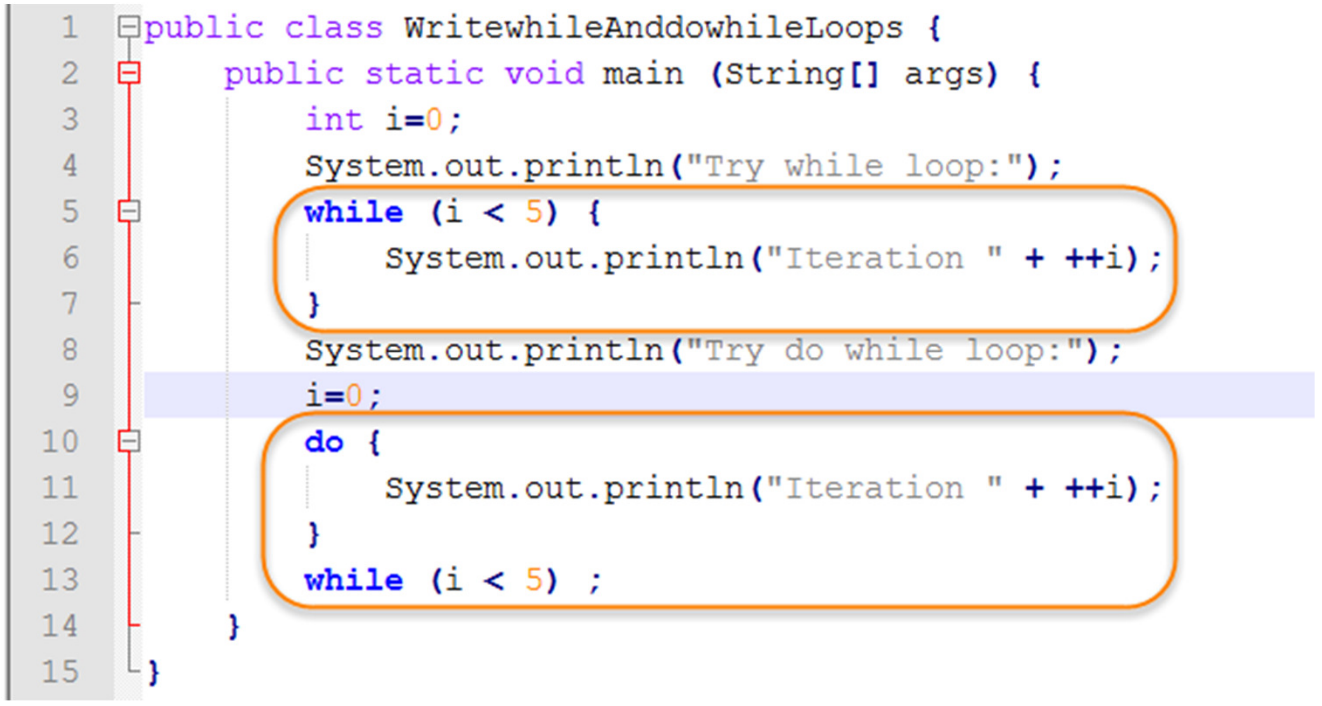
```
for (int i =0; i<10 ; i++) {

// Loop statements to be executed

}
```

Loop Statements



```
public class WriteforEachLoops {
    public static void main (String[] args) {
        String[] names={"Regina","Stephen","Dave","Marsha"};
        System.out.println("For each loop output:");
        for (String name : names) {
            System.out.println(name);
        }
    }
}
```

Loop Statements

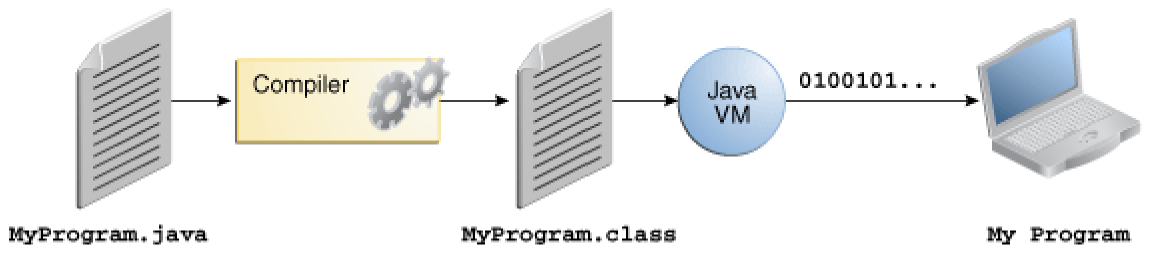


```
public class WritewhileAnddowhileLoops {
    public static void main (String[] args) {
        int i=0;
        System.out.println("Try while loop:");
        while (i < 5) {
            System.out.println("Iteration " + ++i);
        }
        System.out.println("Try do while loop:");
        i=0;
        do {
            System.out.println("Iteration " + ++i);
        }
        while (i < 5) ;
    }
}
```

Compiling, Running and Debugging Java Programs

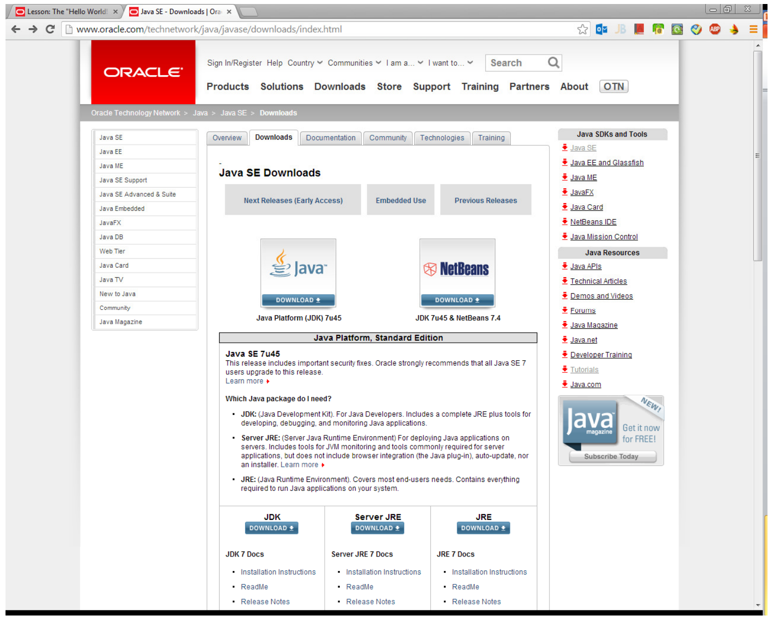
Java Development Process

.java => .class => JVM execution



Installing Java in your machine (1)

- Downloading Java Development Kit (JDK) from [Oracle]
- Java Runtime Environment (JRE) is usually included in the JDK installation file.



Java SE Downloads

Java Platform (JDK) 7u45

JDK 7u45 & NetBeans 7.4

Java Platform, Standard Edition

Java SE 7u45
This release includes important security fixes. Oracle strongly recommends that all Java SE 7 users upgrade to this release.
Learn more

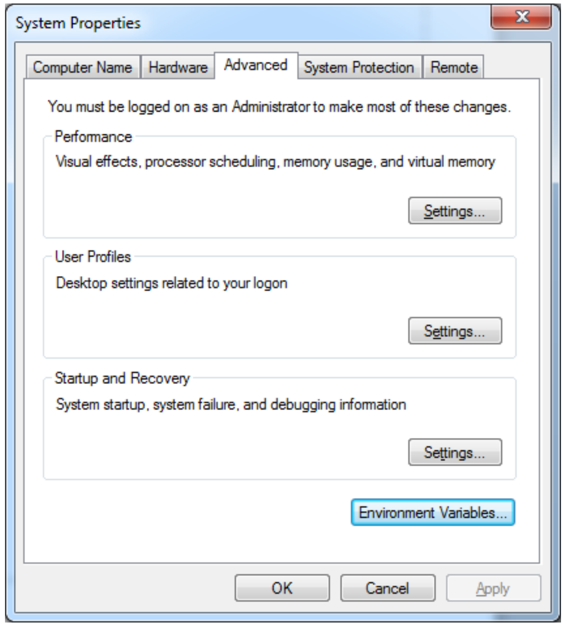
Which Java package do I need?

- **JDK:** (Java Development Kit). For Java Developers. Includes a complete JRE plus tools for developing, debugging, and monitoring Java applications.
- **Server JRE:** (Server Java Runtime Environment) For deploying Java applications on servers. Includes tools for JVM monitoring and tools commonly required for server applications, but does not include browser integration (the Java plug-in), auto-update, nor an installer. Learn more
- **JRE:** (Java Runtime Environment). Covers most end-users needs. Contains everything required to run Java applications on your system.

JDK 7 Docs	Server JRE 7 Docs	JRE 7 Docs
Installation Instructions	Installation Instructions	Installation Instructions
ReadMe	ReadMe	ReadMe
Release Notes	Release Notes	Release Notes

Installing Java in your machine (2)

- Setting JAVA_HOME (Windows):
  - E.g., *C:\Program Files\Java\jdk1.7.0_45*
- Setting **path** and **classpath**

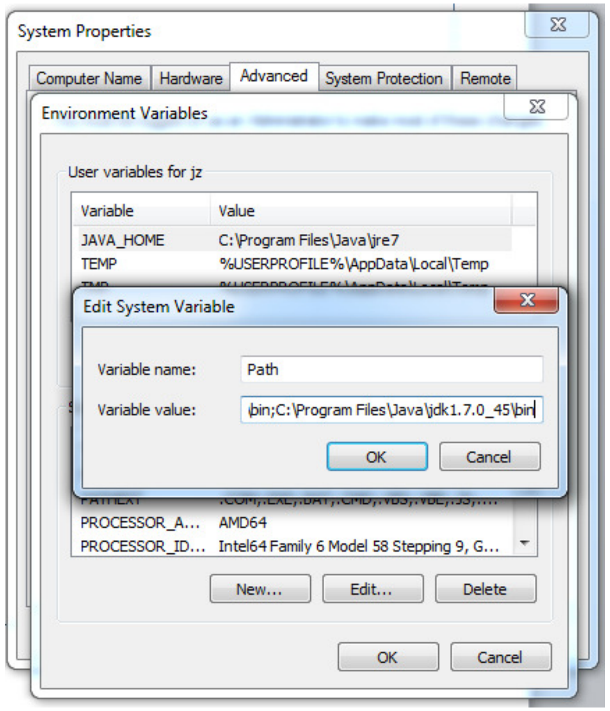

System Properties

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information


System Properties

Environment Variables

User variables for jz

Variable	Value
JAVA_HOME	C:\Program Files\Java\jre7
TEMP	%USERPROFILE%\AppData\Local\Temp

Edit System Variable

Variable name: Path

Variable value: bin;C:\Program Files\Java\jdk1.7.0_45\bin

Variable	Value
PROCESSOR_A...	AMD64
PROCESSOR_ID...	Intel64 Family 6 Model 58 Stepping 9, G...

Compile .java File into a .class File (Command Line)

Linux:

```
javac -cp “. : libs/*” CapitalizeTest.java
```

Windows:

```
javac -cp “. ; libs/*” CapitalizeTest.java
```

Run
(Command Line)

Linux:

```
java -cp “. : libs/*” CapitalizeTest
```

Windows:

```
java -cp “. ; libs/*” CapitalizeTest
```

Demo

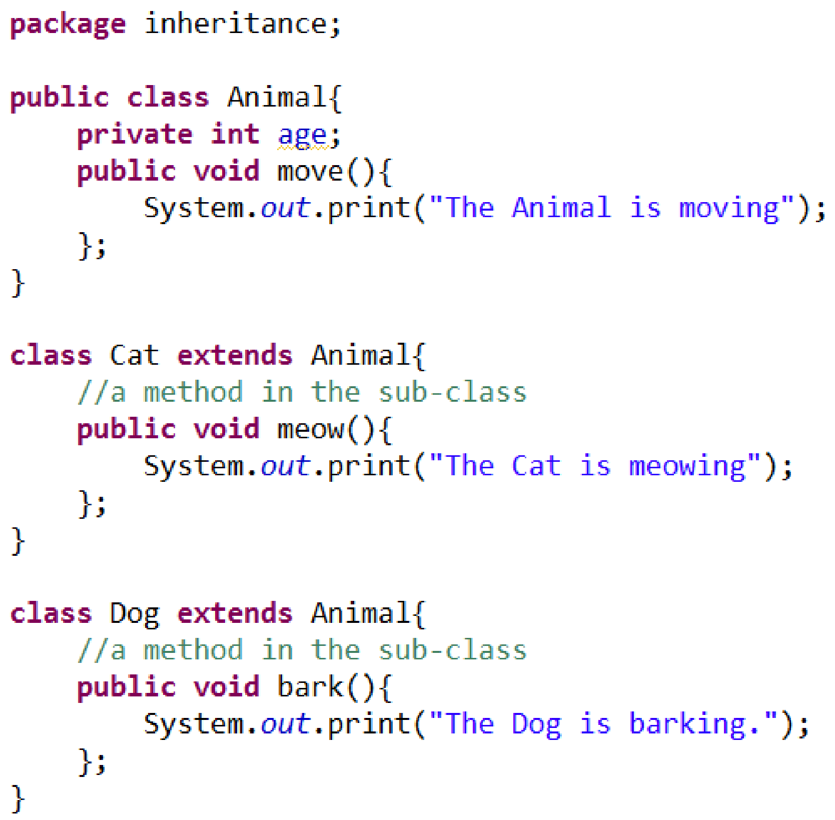
- Download the files
- You need two jar files to compile the code:
  commons-lang3, commons-text

```java
import org.apache.commons.text.WordUtils;

public class CapitalizeTest {

    public static void main(String[] args) {
        System.out.println(WordUtils.capitalize("this is a string"));
    }

}
```

Java Inheritance

Inheritance in Java

- Java classes can be *derived* from other classes, thereby *inheriting* fields and methods from those classes.

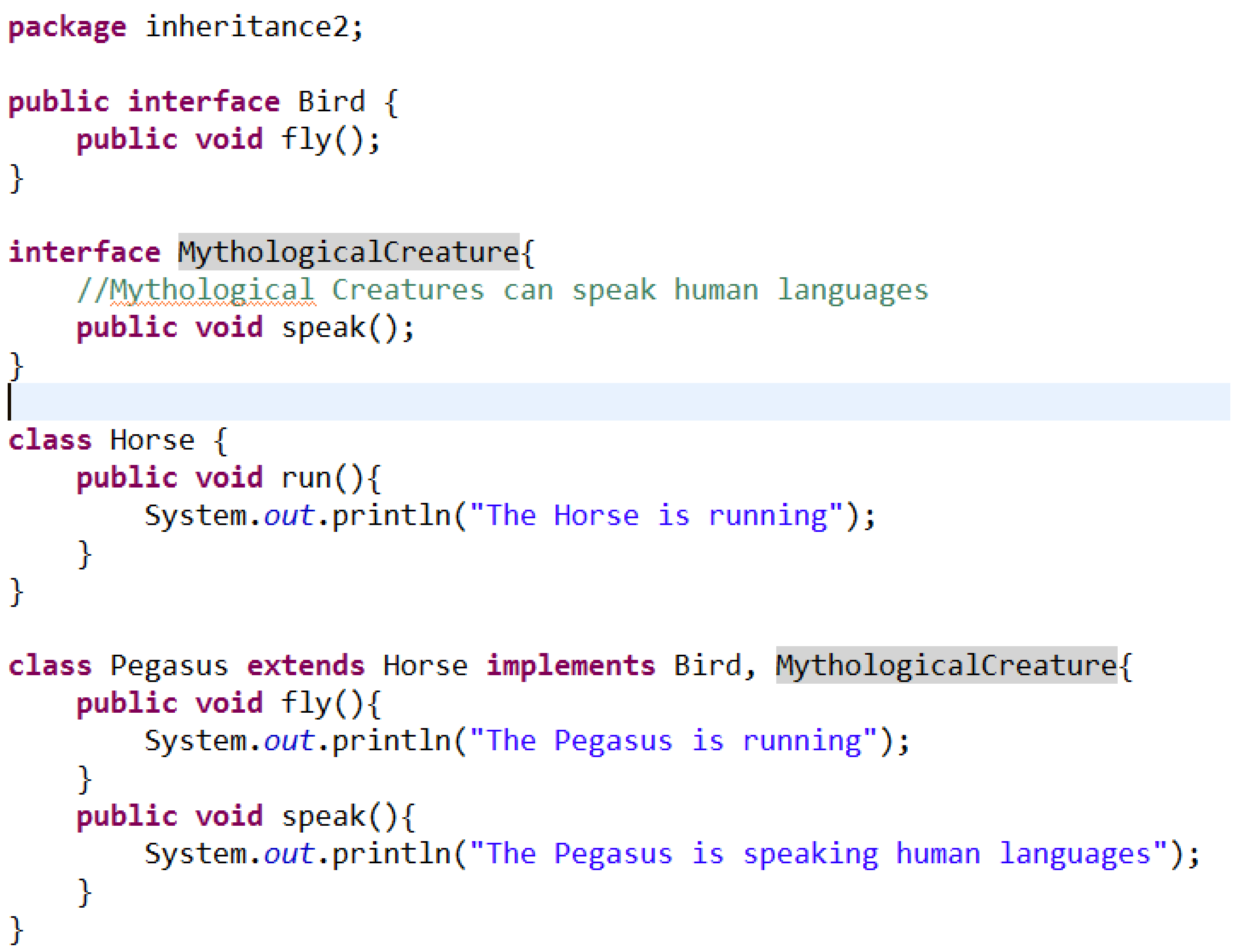


```
package inheritance;

public class Animal{
    private int age;
    public void move(){
        System.out.print("The Animal is moving");
    };
}

class Cat extends Animal{
    //a method in the sub-class
    public void meow(){
        System.out.print("The Cat is meowing");
    };
}

class Dog extends Animal{
    //a method in the sub-class
    public void bark(){
        System.out.print("The Dog is barking.");
    };
}
```

Interface

```
package inheritance2;

public interface Animal {
    public void move();
    public void eat();
}

class Dog implements Animal
{
    public void move() {
        System.out.println("The Dog is moving.");
    }
    public void eat() {
        System.out.println("The Dog is eating.");
    }
}

class Cat implements Animal
{
    public void move() {
        System.out.println("The Dog is moving.");
    }
    public void eat() {
        System.out.println("The Dog is eating.");
    }
}
```

“Multiple Inheritance”

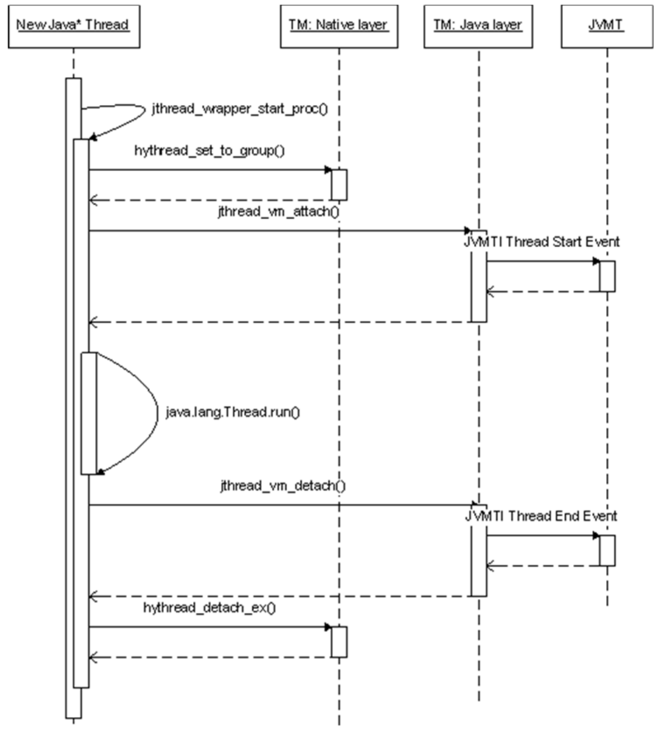


```java
package inheritance2;

public interface Bird {
    public void fly();
}

interface MythologicalCreature{
    //Mythological Creatures can speak human languages
    public void speak();
}

class Horse {
    public void run(){
        System.out.println("The Horse is running");
    }
}

class Pegasus extends Horse implements Bird, MythologicalCreature{
    public void fly(){
        System.out.println("The Pegasus is running");
    }
    public void speak(){
        System.out.println("The Pegasus is speaking human languages");
    }
}
```

Java Threading

Java Threading

- A *thread* is a thread of execution in a program [6]
- JVM allows an application to have multiple threads running concurrently.
- Apache Harmony example:



http://harmony.apache.org/subcomponents/drlvm/TM.html

Two ways to do threading

1. Extends Thread class

```
class PrimeThread extends Thread {
    long minPrime;
    PrimeThread(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

```
PrimeThread p = new PrimeThread(143);
p.start();
```

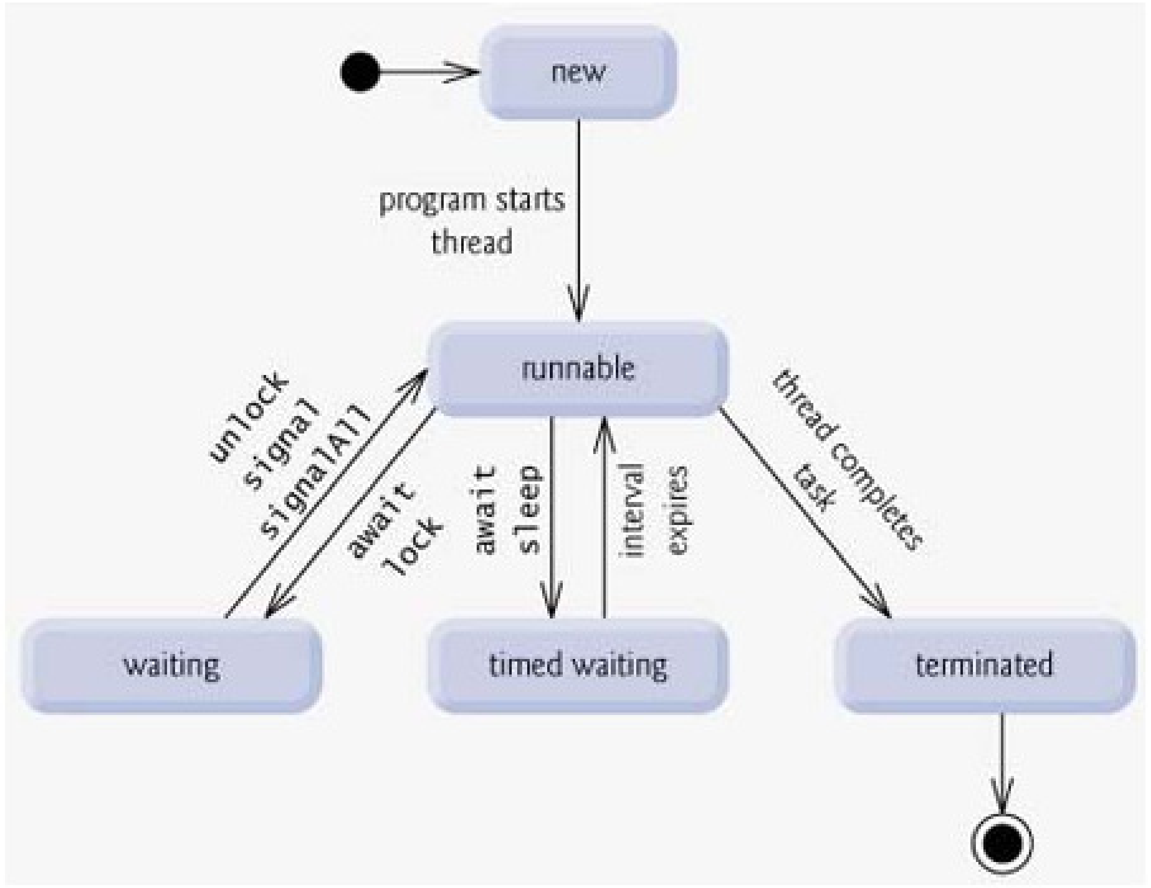
2. Implements Runnable interface

```
class PrimeRun implements Runnable {
    long minPrime;
    PrimeRun(long minPrime) {
        this.minPrime = minPrime;
    }

    public void run() {
        // compute primes larger than minPrime
         . . .
    }
}
```

```
    PrimeRun p = new PrimeRun(143);
    new Thread(p).start();
```

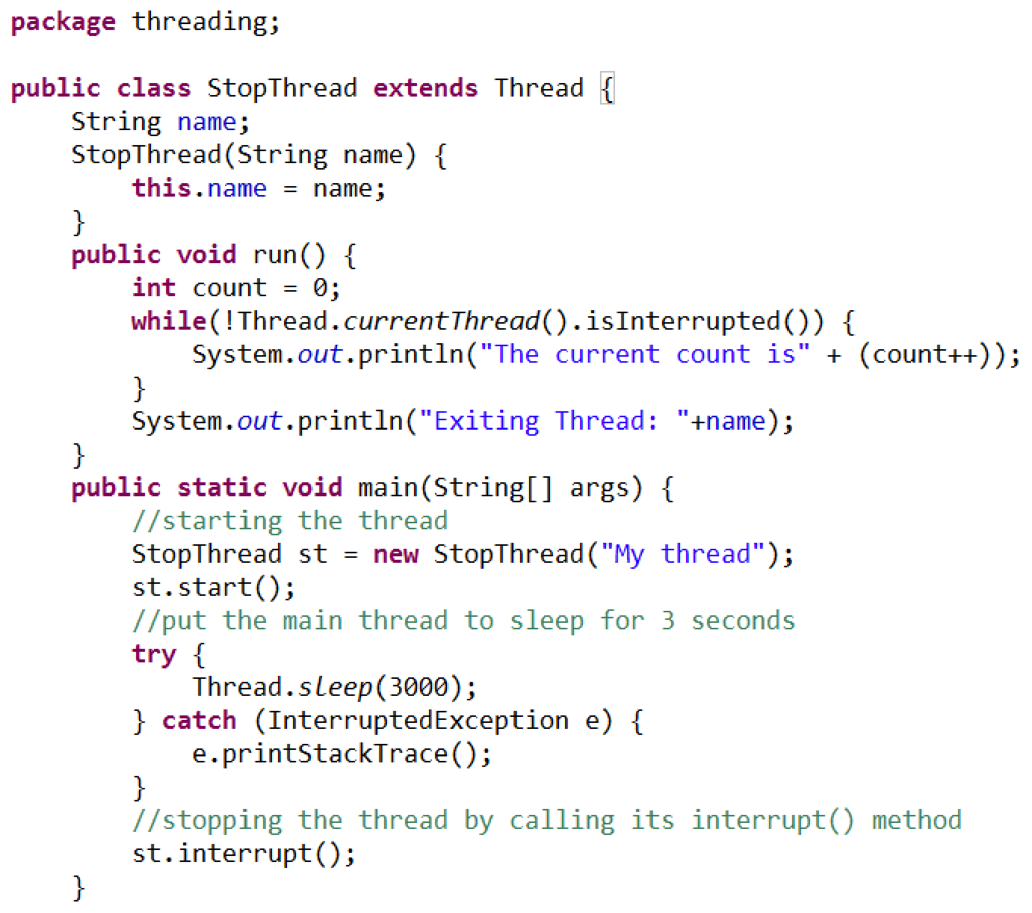
http://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html

Thread lifecycle



How to stop a Thread

- Using Thread.interrupt() method:



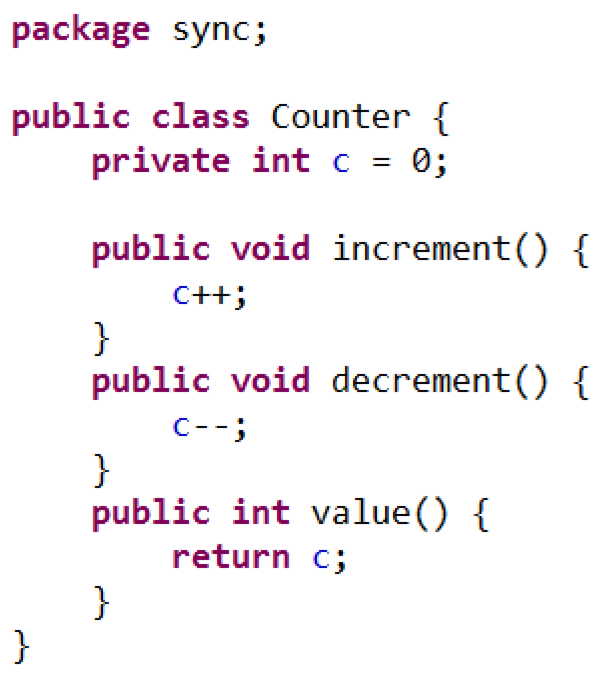
```java
package threading;

public class StopThread extends Thread {
    String name;
    StopThread(String name) {
        this.name = name;
    }
    public void run() {
        int count = 0;
        while(!Thread.currentThread().isInterrupted()) {
            System.out.println("The current count is" + (count++));
        }
        System.out.println("Exiting Thread: "+name);
    }
    public static void main(String[] args) {
        //starting the thread
        StopThread st = new StopThread("My thread");
        st.start();
        //put the main thread to sleep for 3 seconds
        try {
            Thread.sleep(3000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        //stopping the thread by calling its interrupt() method
        st.interrupt();
    }
```

http://stackoverflow.com/questions/7786305/stopping-a-specific-java-thread

Java Synchronization

Thread Interference (1)

- Increment operation is translated to **multiple steps** by the virtual machine :
  1. Retrieve the current value of c.
  2. Increment the retrieved value by 1.
  3. Store the incremented value back in c.



```
package sync;

public class Counter {
    private int c = 0;

    public void increment() {
        c++;
    }
    public void decrement() {
        c--;
    }
    public int value() {
        return c;
    }
}
```

Example from: http://docs.oracle.com/javase/tutorial/essential/concurrency/interfere.html

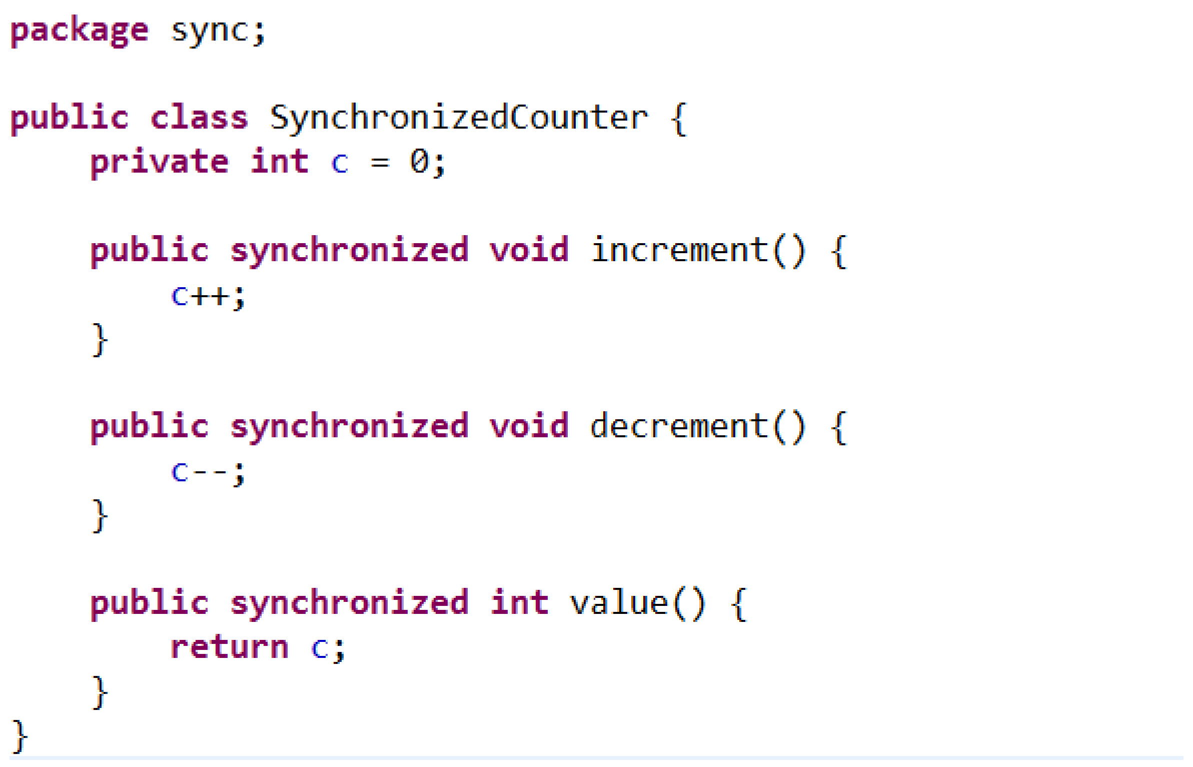
Thread Interference (2)

- Assume we have 2 threads, A and B.
- A increments c, and B decrements c.
- Thread A and B runs together.
- One possible order of the low-level steps:
  1. Thread A: Retrieve c.
  2. Thread B: Retrieve c.
  3. Thread A: Increment retrieved value; result is 1.
  4. Thread B: Decrement retrieved value; result is -1.
  5. Thread A: Store result in c; c is now 1.
  6. Thread B: Store result in c; c is now -1.
- Is the result correct?
- What if the thread A and B are bank transactions?

Problem Root Cause

- Threads are visiting one field (resource) at the same time.
- Multiple steps of an operation
- No enforced “happen-before” relationship

Solution: **synchronized** method



```
package sync;

public class SynchronizedCounter {
    private int c = 0;

    public synchronized void increment() {
        c++;
    }

    public synchronized void decrement() {
        c--;
    }

    public synchronized int value() {
        return c;
    }
}
```

Example: http://docs.oracle.com/javase/tutorial/essential/concurrency/syncmeth.html

synchronized method

- Enforce the ‘happen-before’ relationship in the method level.
- Either one of the below instance will happen. But result is always 0, which is correct.

1. Thread A: Retrieve c.
2. Thread A: Increment retrieved value; result is 1.
3. Thread A: Store result in c; c is now 1.
4. Thread B: Retrieve c.
5. Thread B: Decrement retrieved value; result is 0.
6. Thread B: Store result in c; c is now 0.

OR

1. Thread B: Retrieve c.
2. Thread B: Decrement retrieved value; result is -1.
3. Thread B: Store result in c; c is now -1.
4. Thread A: Retrieve c.
5. Thread A: Increment retrieved value; result is 0.
6. Thread A: Store result in c; c is now 0.

synchronized statement

- Correct way of using locks: using **new** to instantiate an object

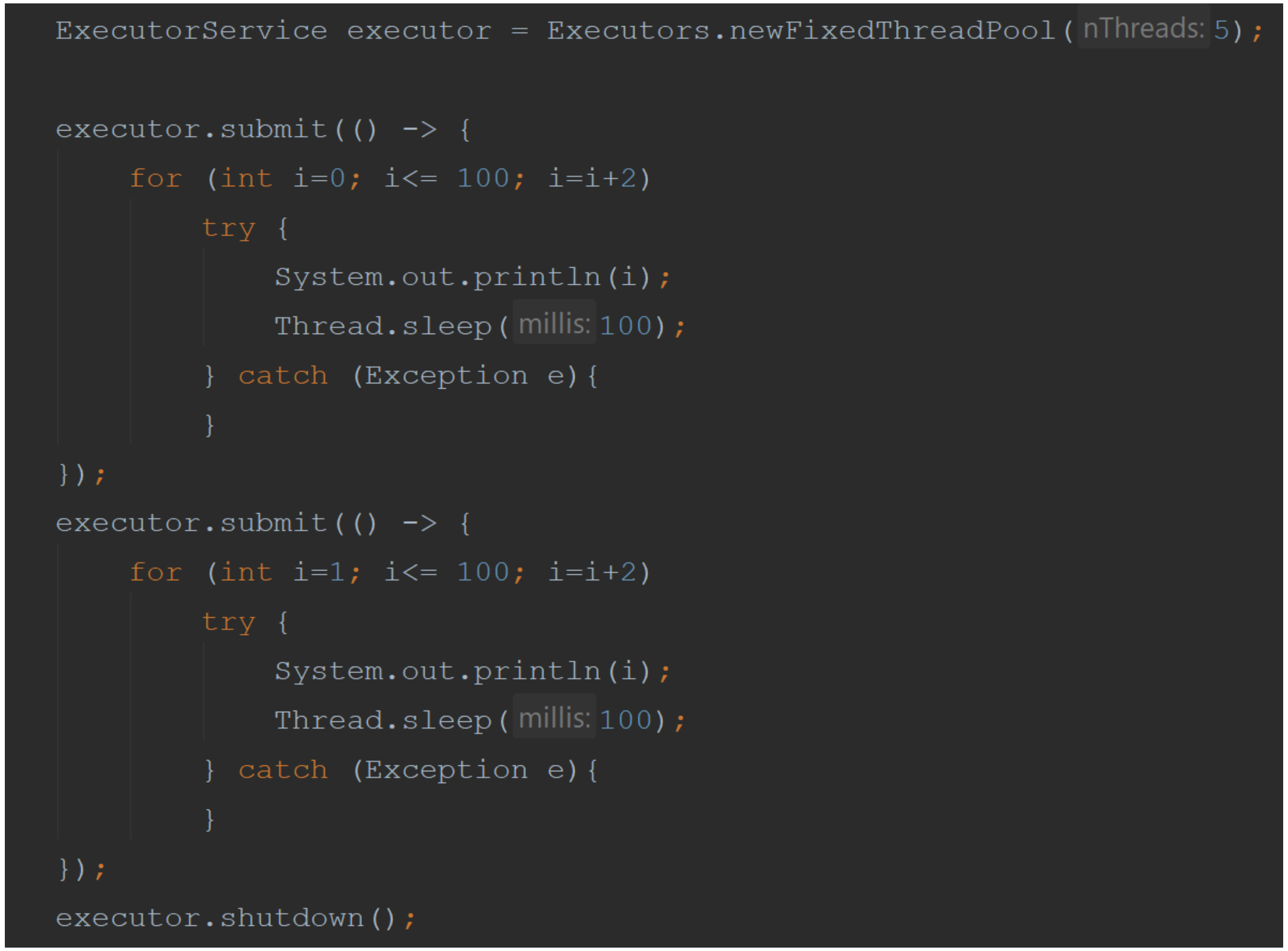
```
int lock = 0;
private final Integer Lock = new Integer(lock);

public void doSomething() {
  synchronized (Lock) {
    // ...
  }
}
```

https://www.securecoding.cert.org/confluence/display/java/LCK01-J.+Do+not+synchronize+on+objects+that+may+be+reused

synchronized statements

- Every object has an intrinsic lock associated with it
- Primitive types (e.g., int, char) do not have intrinsic locks.
- We can combine object intrinsic locks and **synchronized** keyword to create fine-grained synchronization control.

Demo



```
ExecutorService executor = Executors.newFixedThreadPool( nThreads: 5);

executor.submit(() -> {
    for (int i=0; i<= 100; i=i+2)
        try {
            System.out.println(i);
            Thread.sleep( millis: 100);
        } catch (Exception e){
        }
});
executor.submit(() -> {
    for (int i=1; i<= 100; i=i+2)
        try {
            System.out.println(i);
            Thread.sleep( millis: 100);
        } catch (Exception e){
        }
});
executor.shutdown();
```

References

1. Thinking in Java 4th Ed, Bruce Eckel
2. Oracle Java tutorial (http://docs.oracle.com/javase/tutorial/index.html)
3. www.cs.drexel.edu/~spiros/teaching/CS575/**slides/java**.**ppt**
4. http://eclipsetutorial.sourceforge.net/Total_Beginner_Companion_Document.pdf
5. http://www.vogella.com/tutorials/EclipseDebugging/article.html