

INTRODUCTION TO JAVA LANGUAGE

BEGINNERS TO ADVANCE



GENIAL  ODE

www.genial-code.com

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**
 - Distributed
 - Multithreaded
 - Dynamic
 - **Architecture neutral**
 - Portable
 - High performance
 - Robust
 - Secure



EXAMPLE: HELLO WORLD PROGRAM

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

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



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() ;
```



COMPARED WITH C/C++ [3]

- Java has no:
 - pointers
 - typedef
 - preprocessor
 - struct
 - unions
 - multiple inheritance
 - goto
 - operator overloading
 - malloc
 - ...



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);
    }
}
```

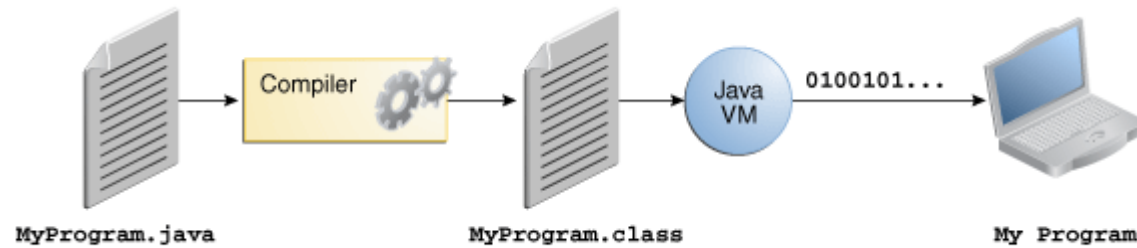


COMPILING, RUNNING AND DEBUGGING JAVA PROGRAMS

10

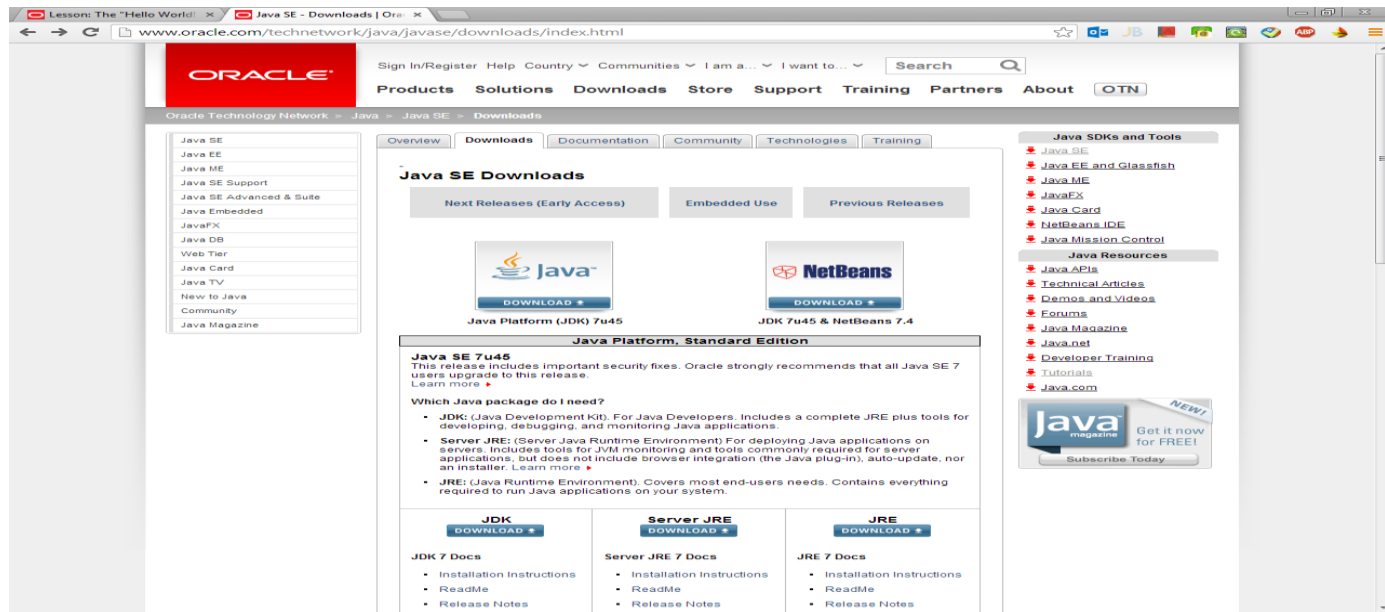
JAVA DEVELOPMENT PROCESS

.java => .class => JVM execution



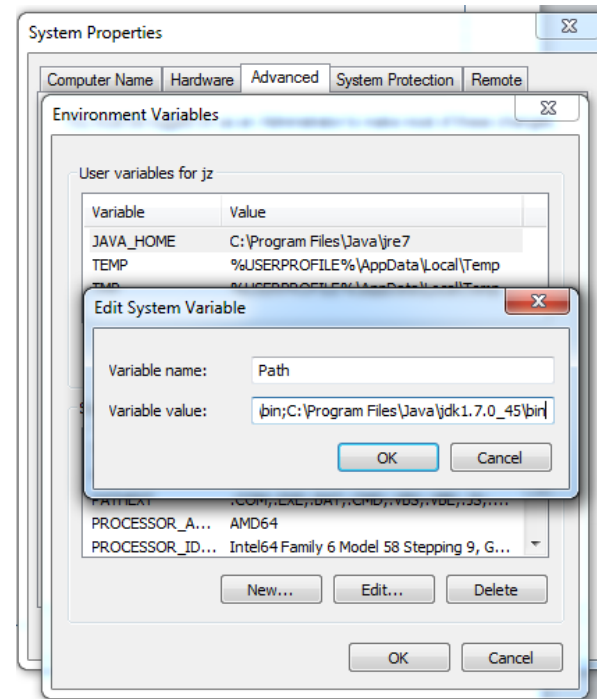
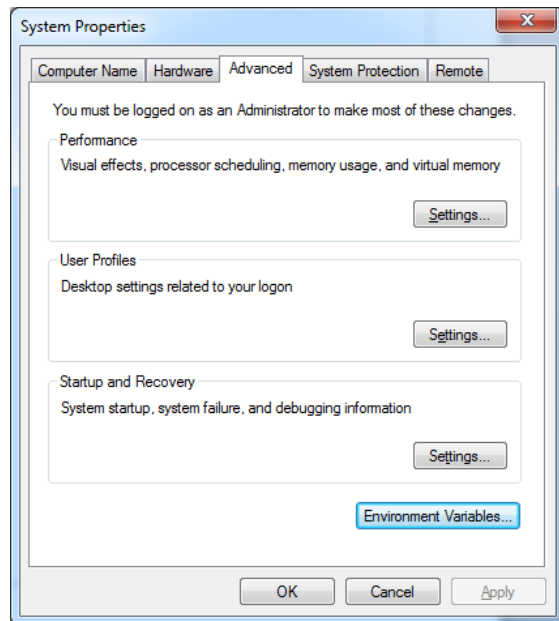
INSTALLING JAVA IN YOUR MACHINE (1)

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



INSTALLING JAVA IN YOUR MACHINE (2)

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



COMPILE .JAVA FILE INTO A .CLASS FILE (COMMAND LINE)

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\jz>cd C:\Users\jz\Documents\java_files

C:\Users\jz\Documents\java_files>dir
Volume in drive C is ??
Volume Serial Number is 000E-81F4

Directory of C:\Users\jz\Documents\java_files
07/01/2014  11:45 AM    <DIR>          .
07/01/2014  11:45 AM    <DIR>          ..
07/01/2014  11:45 AM                426 HelloWorld.class
07/01/2014  11:45 AM                115 HelloWorld.java
                2 File(s)                541 bytes
                2 Dir(s)  13,493,460,992 bytes free

C:\Users\jz\Documents\java_files>javac HelloWorld.java

C:\Users\jz\Documents\java_files>set classpath=.

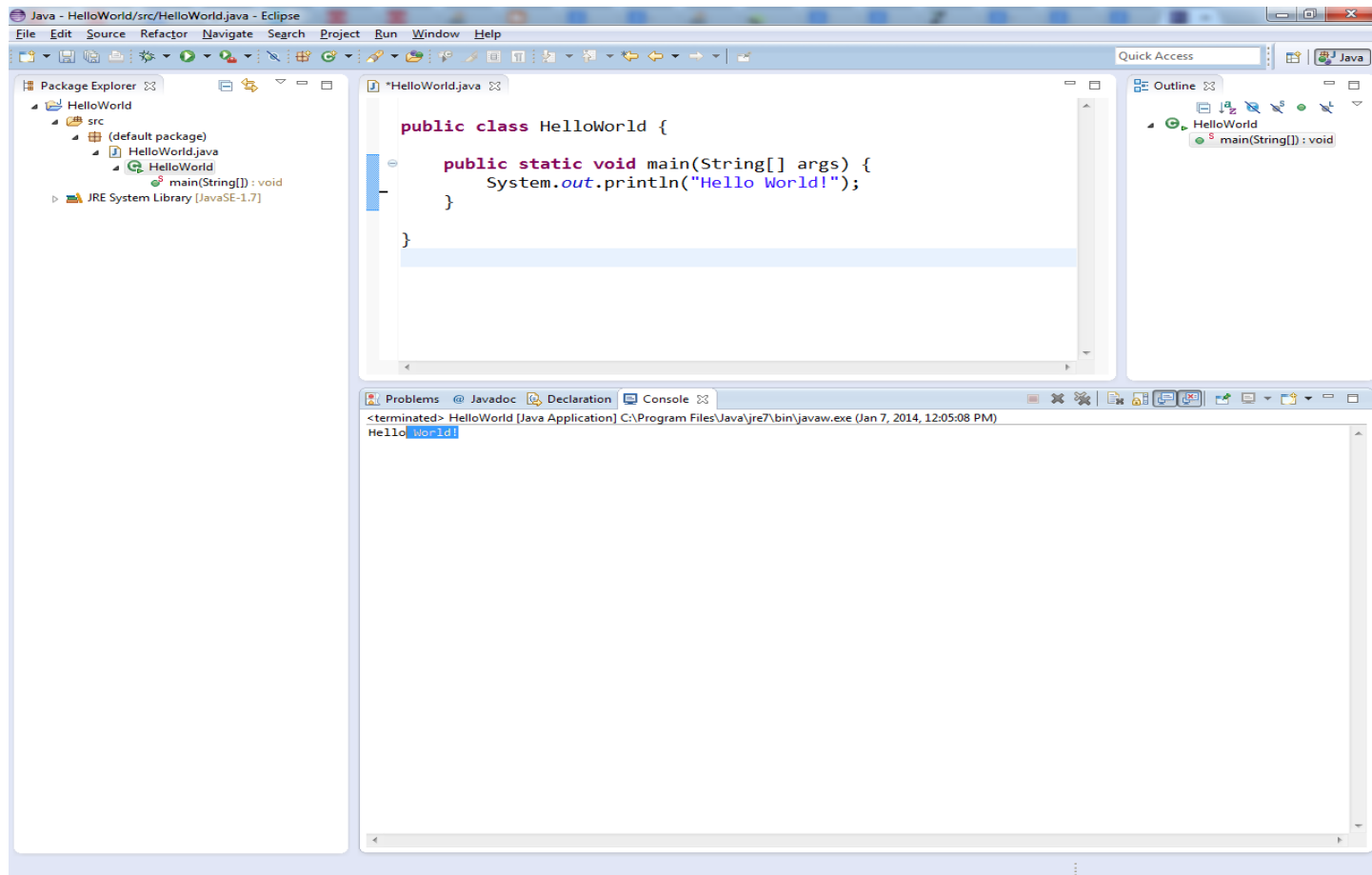
C:\Users\jz\Documents\java_files>java HelloWorld
Hello World!

C:\Users\jz\Documents\java_files>
```

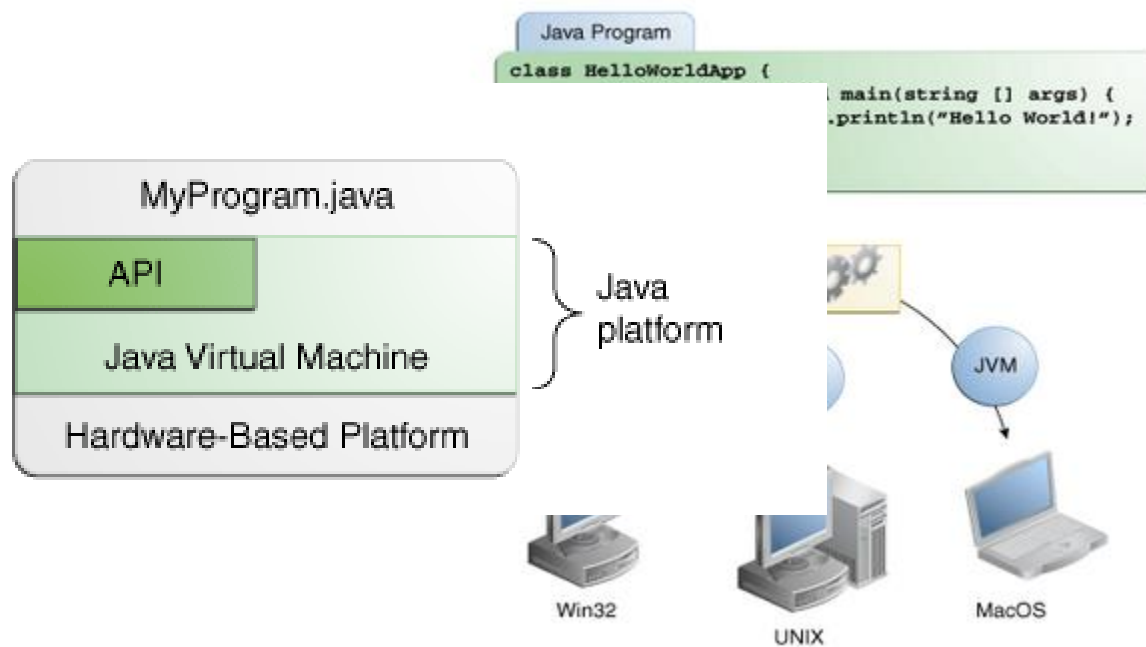


RUNNING HELLOWORLD IN ECLIPSE IDE

Eclipse Download from [here](#).



JAVA PLATFORM



DEBUGGING JAVA IN ECLIPSE (1)

- ***Debugging*** means “run a program interactively while watching the source code and the variables during the execution.” [5]
- Set ***breakpoints*** to stop the program at the middle of execution
- Eclipse has a *Debug Mode*

DEBUGGING JAVA IN ECLIPSE(2)

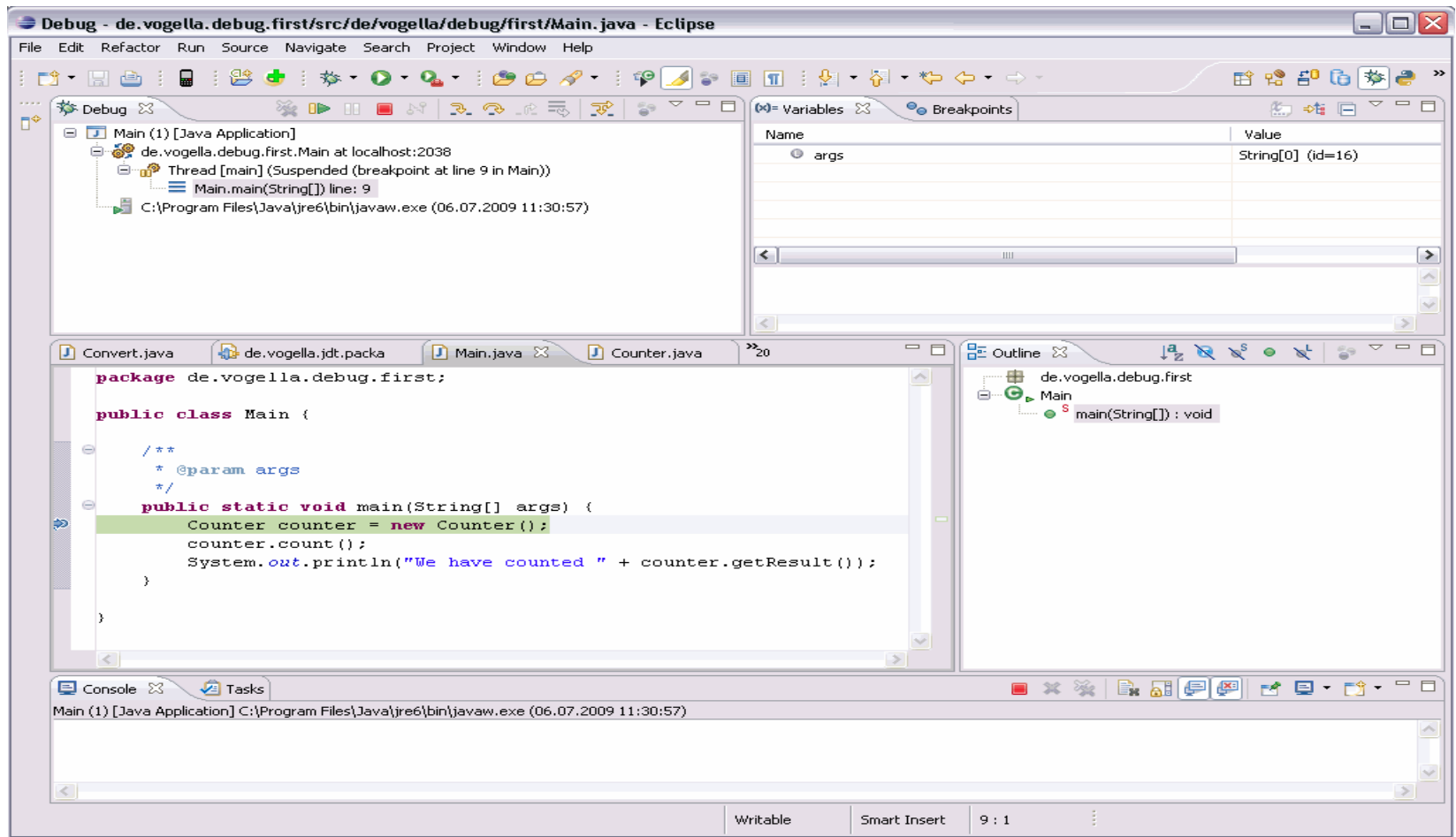


Image courtesy:

<http://www.vogella.com/tutorials/EclipseDebugging/images/xdebugstart20.gif.pagespeed.ic.SqCELlNeCm.png>

DEBUGGING JAVA IN ECLIPSE(3)



Table 1. Debugging key bindings / shortcuts

Key	Description
F5	Executes the currently selected line and goes to the next line in your program. If the selected line is a method call the debugger steps into the associated code.
F6	F6 steps over the call, i.e. it executes a method without stepping into it in the debugger.
F7	F7 steps out to the caller of the currently executed method. This finishes the execution of the current method and returns to the caller of this method.
F8	F8 tells the Eclipse debugger to resume the execution of the program code until it reaches the next breakpoint or watchpoint.

Table courtesy: <http://www.vogella.com/tutorials/EclipseDebugging/article.html>

JAVA INHERITANCE

20

INHERITANCE IN JAVA

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

```
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.");
    };
}
```



```

graph TD
    Object[Object] --> C1[...]
    Object --> C2[...]
    Object --> C3[...]
    Object --> C4[...]
    Object --> C5[...]
    Object --> C6[...]
    C1 --> C1_1[...]
    C1 --> C1_2[...]
    C1 --> C1_3[...]
    C2 --> C2_1[...]
    C2 --> C2_2[...]
    C3 --> C3_1[...]
    C3 --> C3_2[...]
    C4 --> C4_1[...]
    C4 --> C4_2[...]
    C5 --> C5_1[...]
    C5 --> C5_2[...]
    C5 --> C5_3[...]
    C6 --> C6_1[...]
    C6 --> C6_2[...]
    C6 --> C6_3[...]
  
```



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”

```
package inheritance2;

public interface Bird {
    public void fly();
}

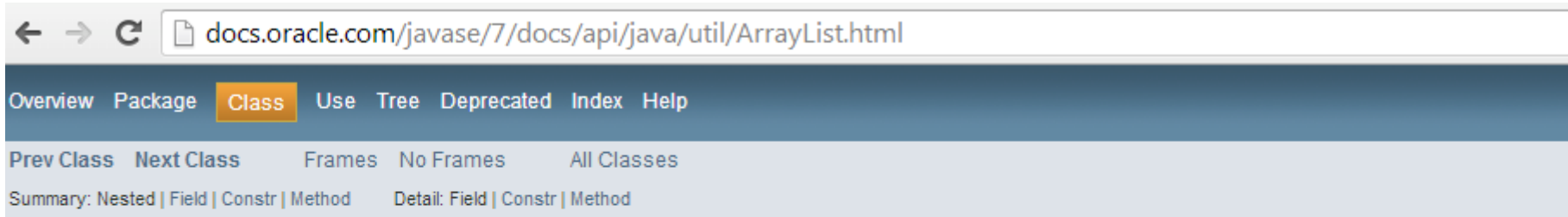
interface MythologicalCreature{
    //Mythological Creatures can speak human languages
    public void speak();
}

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");
    }
}
```



A REAL WORLD EXAMPLE: ARRAYLIST



java.util

Class ArrayList<E>

java.lang.Object

java.util.AbstractCollection<E>

java.util.AbstractList<E>

java.util.ArrayList<E>

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
    extends AbstractList<E>
    implements List<E>, RandomAccess, Cloneable, Serializable
```

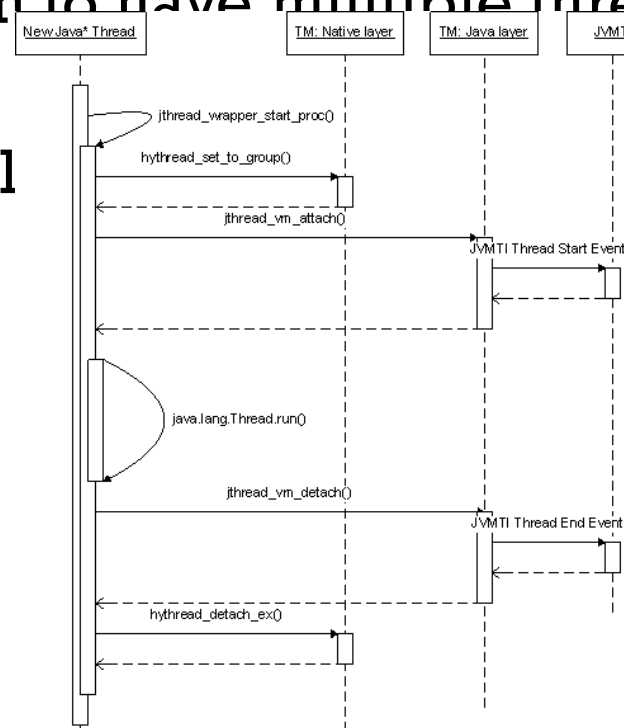


JAVA THREADING

26

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



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();
```

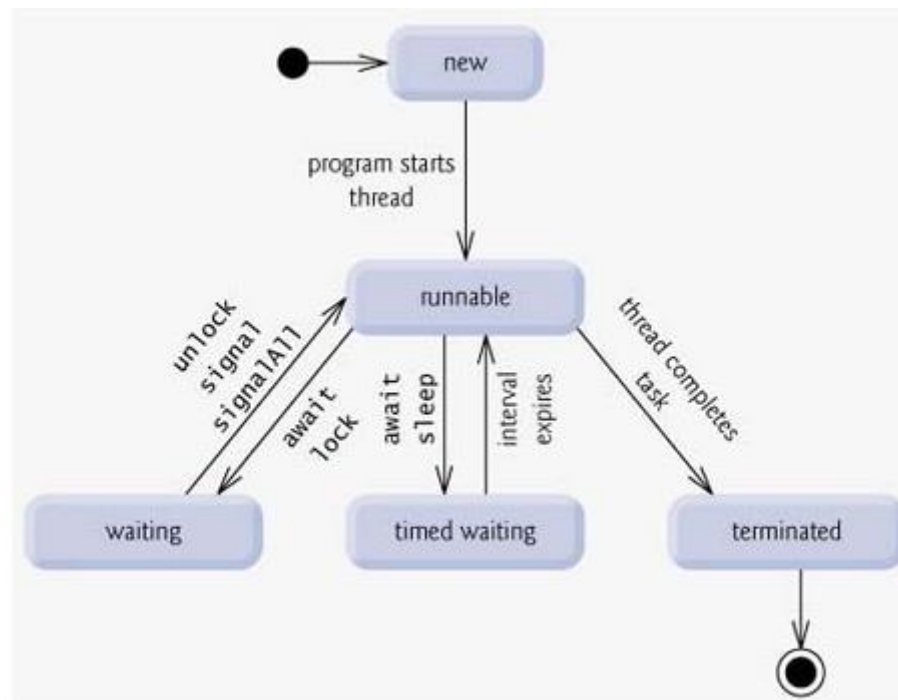
2. Implements Runnable interface

```
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();
```



THREAD LIFECYCLE



HOW TO STOP A THREAD

■ Using `Thread`

```
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();
    }
}
```



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;
    }
}
```



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;
    }
}
```



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. **OR**
5. Thread B: Decrement retrieved value; result is 0.
6. Thread B: Store result in c; **c is now 0.**

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 STATEMENTS (1)

- 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.



SYNCHRONIZED STATEMENTS (2)

```
package sync;

public class MsLunch {
    private long c1 = 0;
    private long c2 = 0;

    //each object has its own intrinsic lock
    private Object lock1 = new Object();
    private Object lock2 = new Object();

    public void inc1() {
        //combine object intrinsic locks and synchronized keyword
        synchronized(lock1) {
            c1++;
        }
    }

    public void inc2() {
        synchronized(lock2) {
            c2++;
        }
    }
}
```



SYNCHRONIZED STATEMENT HAZARDS

(1)

```
private final Boolean initialized = Boolean.FALSE;

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

- Boolean has only two instances of Boolean
- If another thread also synchronizes on the same Boolean instance, like this:
 - private final Boolean **someLock** = Boolean.FALSE;
- The lock will be reused.
- The system might be deadlock or unresponsive.
- It is hard to detect this type of bugs!

More examples:

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



SYNCHRONIZED STATEMENT HAZARDS

- (2)
- Another example of the wrong way of using locks:

```
int lock = 0;
private final Integer Lock = lock; // Boxed primitive Lock is shared

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

What will happen another thread also synchronizes on an integer instance with the 0 integer value?

SYNCHRONIZED STATEMENT HAZARDS

(3)

- 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) {
        // ...
    }
}
```



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>

