

# Introduction to Java™



## Module 1: Getting started, Java Basics

# Lab Objectives



## o Objective:

- Learn how to write, compile and execute HelloWorld.java
- Learn how to use the Java compiler (javac) and interpreter (java).

## o Practice:

- You will write, compile and execute your first Java program
  - ✓ You will write your program using emacs
  - ✓ You will call the Java compiler and interpreter through the command line of a Unix shell.

# What is Java?



Programming language developed by Sun Microsystems

- Simple
- Object Oriented
- Distributed
- Robust
- Secure
- Architecture Neutral
- Portable
- Interpreted
- High Performance
- Multithreaded
- Dynamic
- Network savvy

# Why Java?

- The Java Language has many good design features - secure, safe (with respect to bugs), object-oriented, familiar (to C C++ and even Fortran programmers).
- Good set of libraries: networking, multimedia, from graphics to math functions.
- Best available electronic and paper training resources.
- Children will learn Java as it is a social language with natural graphical "hello world".

# Why Java?

- Java is rapidly getting the better (the best!) Integrated Development Environments (IDEs) for programs.
- Java is naturally integrated with network and universal machine supports potentially powerful "write once-run anywhere" model.
- Easy to teach - I use it to help students understand client/server programming and concurrency - (dining philosophers).
- There is a large and growing trained labour force.

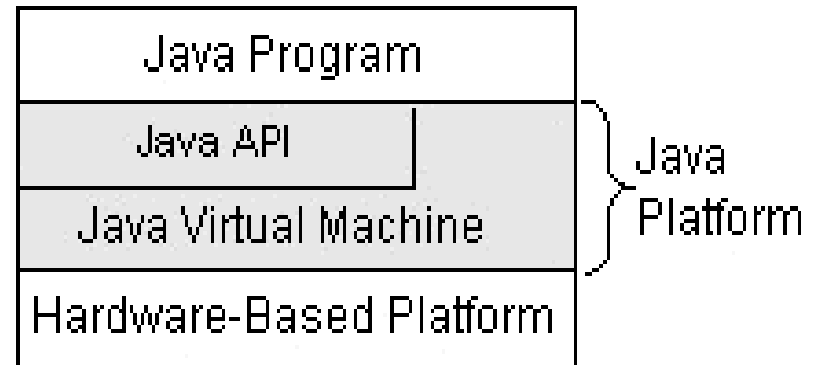
# Java: Key Technical Ideas



- A Better Language:
  - Simplicity and C/C++ compatibility promote fluency;
  - GC and Threads allow software components;
  - Platform independence saves time;
  - Strong typing catches errors up front;
  - Declared exceptions forces coverage in code.
- Scalable Applications:
  - Threads for parallel speedup; patterns “in the large”.
  - Dynamic linking allows simple apps to grow;
  - Range of implementations from JavaCard to HotSpot.

# The Java platform

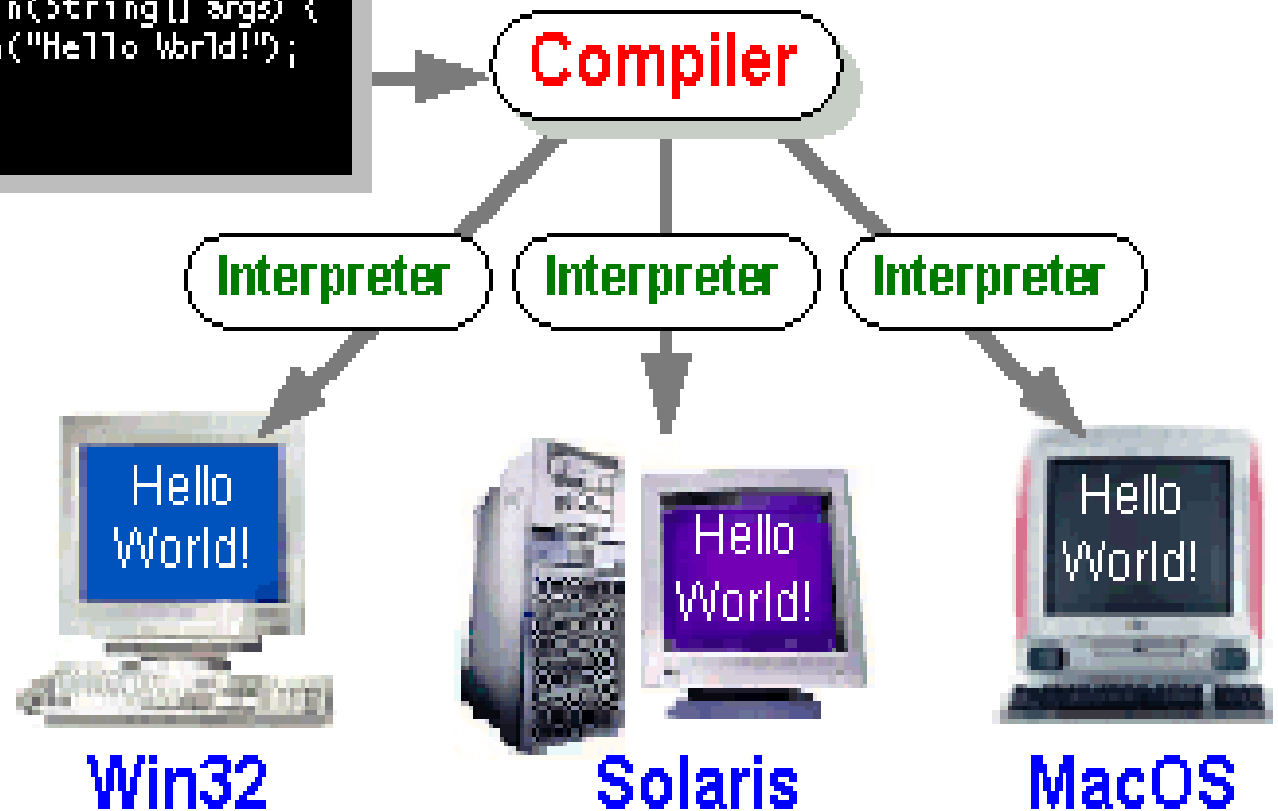
- The execution environment of Java programs
  - Java VM is platform **dependent**
  - Java API is platform **independent**

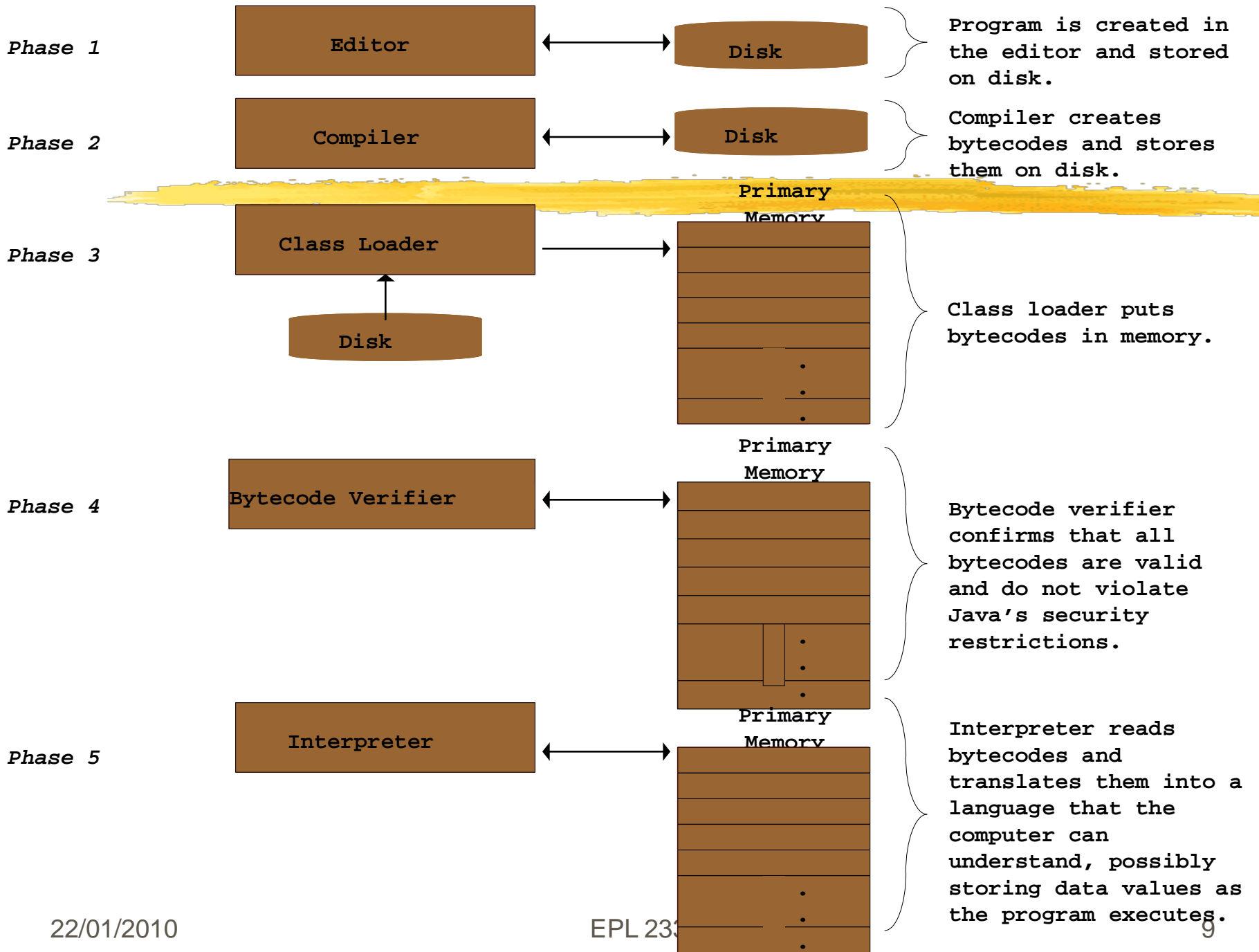


# Java Program

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

HelloWorldApp.java





# Garbage collection



- Automatic Garbage Collection
  - Performed by the Java VM
  - Objects that are not referenced are removed
- To “destruct” an object manually make it equal to **null**. It will be collected by the Garbage Collector
- The Garbage Collector releases memory allocated with the **new** keyword (constructor call)

# Classes and Packages



## o Class

- Defines an object
- The same as in C++

## o Package

- Directory of classes
- As libraries in C++

# Java Documentation



- Can be downloaded from Sun and installed on the local computer
- Can be found online at <http://java.sun.com/>
- Contain the properties and methods of every object as well as the inheritance tree

# Syntax & common commands

- Java syntax is very similar to that of C/C++
- Very common commands:
  - `extends` → inherits from (no multiple inheritance)
  - `System.out.println();` → print to standard output
  - `import` → include
  - How `import` works:
    - ✓ `import java.awt.*;` → will include all subclasses of the `java.awt` package with their methods
    - ✓ `import java.awt.Button;` → will include only the methods of the `java.awt.Button` class

# Java Syntax Tips

- Variables can be declared **anywhere** in the code.
  - They have to be initialized before use
  - Their scope is only within the code fragment enclosed by { } in which they were declared.

Example: `for(int i=0;i<10;i++)`  
`{ ← scope of i → }`

- `Foo x;`  
`x = new Foo();`  
is equivalent to:  
`Foo x = new Foo();`

# Program Syntax



- o General Java Syntax:

- o Class declaration:

```
public class myButton extends Button{  
    //variables, constructors and methods go here  
}
```

public, private static, abstract, final class myButton extends Button implements interface

- o Method declaration:

```
public void changeColor(Color x) {  
    //code goes here  
}
```

public, private void, int, Integer, String ... changeColor

# Program Syntax



- o Constructor declaration:

```
public myButton() {  
  //code to initialize the button  
}
```

- No return type in the constructor.
  - The constructor is usually public, but not necessarily
- o In a Java application (not applet) the **main** method must be declared. This is the first method executed when the program starts
- o The syntax of the **main** method cannot be changed:  

```
public static void main(String[] args) {...}
```

**args** is an array containing the command line arguments

# Program Syntax



## o General program syntax:

```
import java.awt.*;
```

```
public class myButton extends Button{  
    int a;  
    String str;
```

```
    public myButton() { //constructor  
                        //construct the object  
    }
```

```
    public void changeColor(){ //a method  
        this.setColor(Color.red);  
    }
```

```
} //end class
```

# Primitive types



- o There are 8 primitive types in Java.
  - int vs Integer
  - float vs Float
  - double vs Double
  - long vs Long
  - short vs Short
  - char vs Character
  - boolean vs Boolean
  - byte vs Byte

# Creating a Java program

## ○ Required software (minimum):

- Java Development Kit for your platform (includes the Java compiler, Java Virtual Machine, Appletviewer, Libraries and other utilities)
- A text editor

## ○ Recommended software:

- An IDE (Integrated Development Environment)
  - ✓ There are many IDEs available (e.g. NetBeans, Eclipse, etc.)
  - ✓ We will use Eclipse

## ○ The **CLASSPATH** variable

- Tells the Java compiler and Java virtual machine where the libraries are stored
- It should include the current directory (.)
- Usually set in a batch file for easy execution (more on this in a later lab)

# Creating Your First Application - HelloWorldApp



To create this program, you will:

- o Create a Java source file (\*.java).
- o Compile the source file into a bytecode file. The Java compiler, `javac`, takes your source file and translates its text into instructions that the *Java Virtual Machine* (Java VM) can understand.
- o Run the program contained in the bytecode file. The Java VM is implemented by a Java interpreter, `java`. This interpreter takes your bytecode file and carries out the instructions by translating them into instructions that your computer can understand.

# Create a Java Source File



- If you are **NOT USING an IDE**:
  - The Java files you create should be kept in a separate directory (with mkdir).
  - Start a text editor.
  - Type the code and store it in a file HelloWorldApp.java
- If you are **USING an IDE**:
  - Each IDE has its own procedures
  - Generally you create some sort of project and you add your classes (code) under that project
  - We will demonstrate this with Eclipse

# HelloWorldApp.java

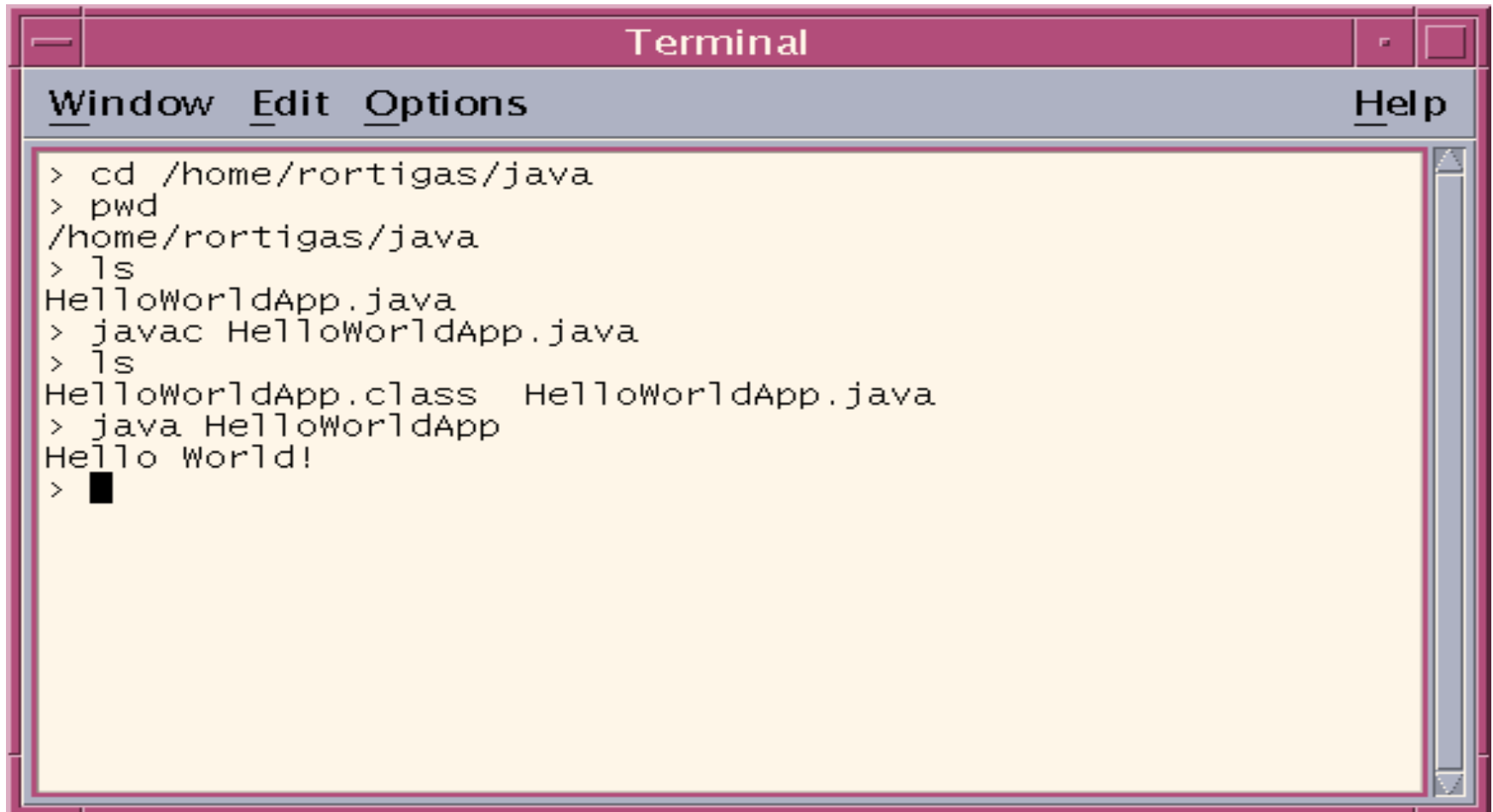
```
/**
 * The HelloWorldApp class implements
 * an application that simply displays
 * "Hello World!" to the standard output.
 */
public class HelloWorldApp {
    public static void main(String[] args){
        System.out.println("Hello World!");
    }
}
```

# Compile the Source File

- If you are **NOT USING** an IDE:
  - `javac HelloWorldApp.java`
  - If your prompt reappears without error messages, congratulations. You have successfully compiled your program.
  - A `HelloWorldApp.class` file is created
- If you are **USING** an IDE:
  - Depends on your IDE
  - Eclipse does background compiling as you write your code
    - ✓ Possible to instruct full code rebuild
  - Most (if not all) IDEs allow you to export your `.java` and `.class` files

# Run the Program (from a shell)

- o `java HelloWorldApp`



```
Terminal
Window Edit Options Help
> cd /home/rortigas/java
> pwd
/home/rortigas/java
> ls
HelloWorldApp.java
> javac HelloWorldApp.java
> ls
HelloWorldApp.class HelloWorldApp.java
> java HelloWorldApp
Hello World!
> █
```