

Java

Introduction

History of Java

- Java was originally developed by Sun Microsystems starting in 1991
 - James Gosling
 - Patrick Naughton
 - Chris Warth
 - Ed Frank
 - Mike Sheridan
- This language was initially called ***Oak***
- Renamed ***Java*** in 1995

What is Java

- A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language
-- Sun
- Object-Oriented
 - No free functions
 - All code belong to some class
 - Classes are in turn arranged in a hierarchy or package structure

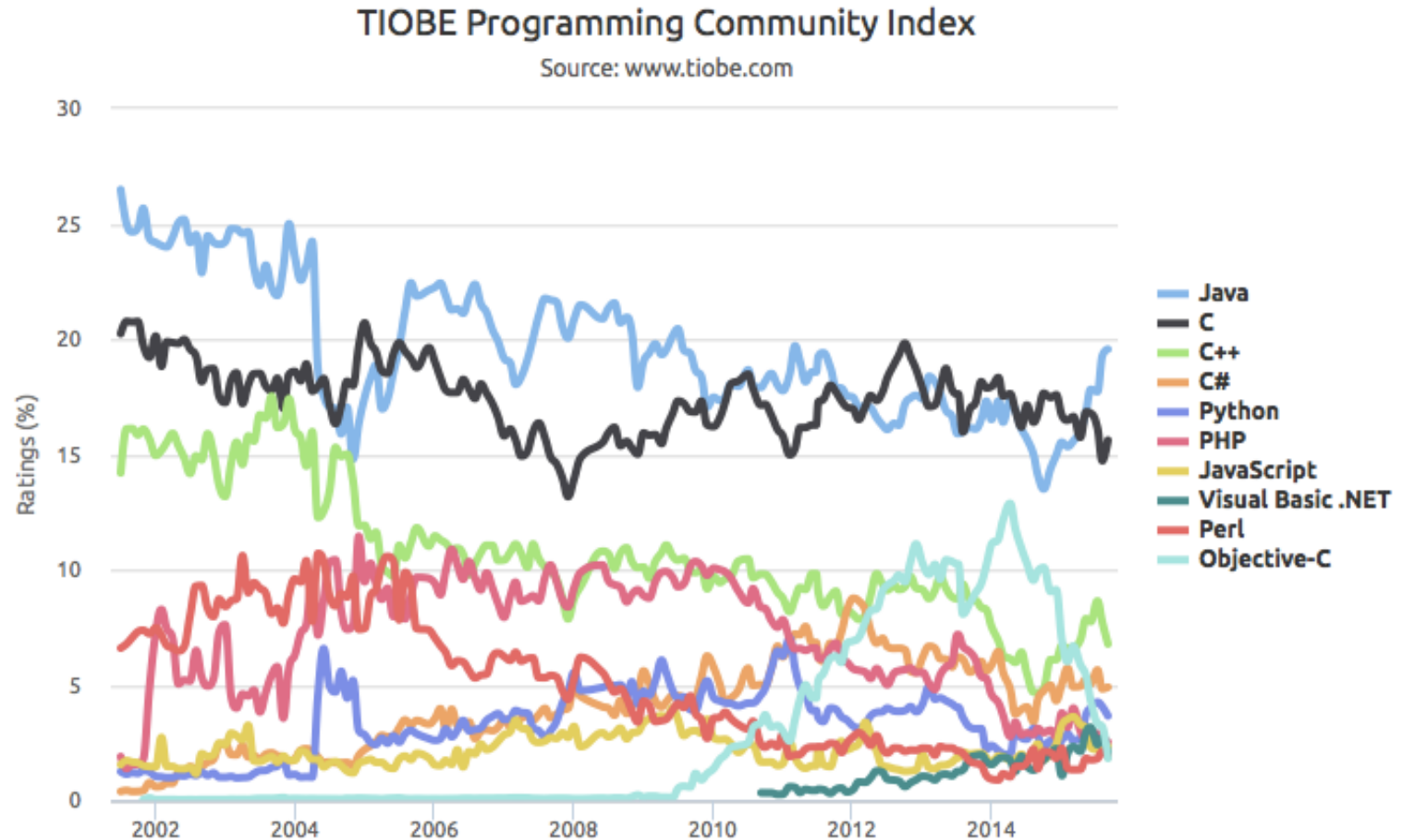
What is Java

- Distributed
 - Fully supports IPv4, with structures to support IPv6
 - Includes support for Applets: small programs embedded in HTML documents
- Interpreted
 - The program are compiled into Java Virtual Machine (JVM) code called bytecode
 - Each bytecode instruction is translated into machine code at the time of execution

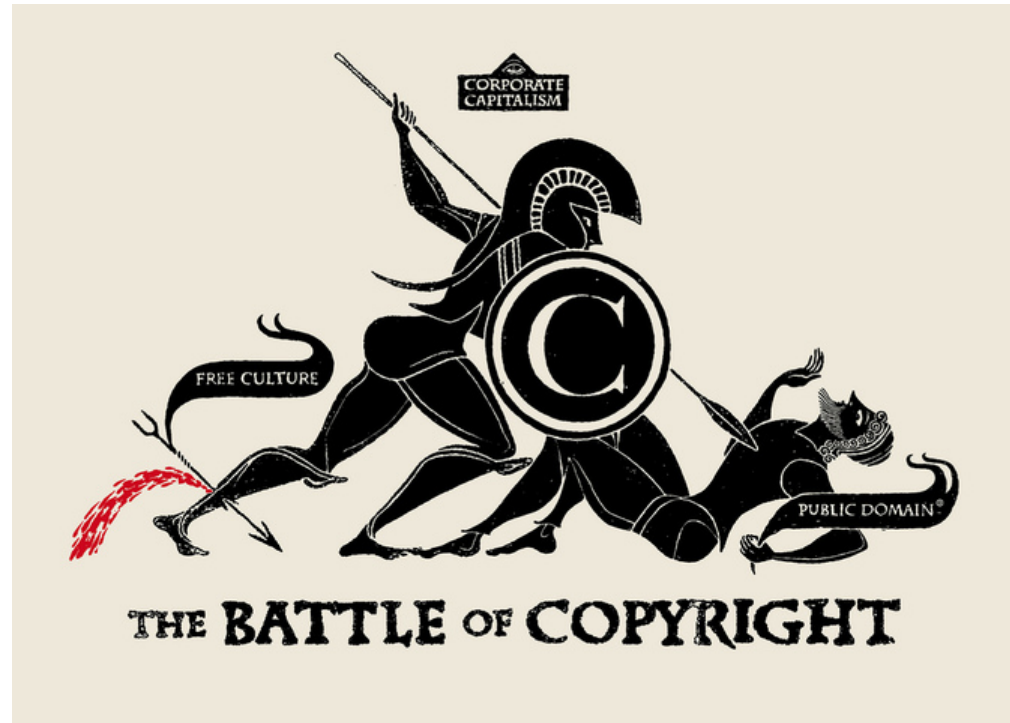
What is Java

- Robust
 - Java is simple – no pointers/stack concerns
 - Exception handling – try/catch/finally series allows for simplified error recovery
 - Strongly typed language – many errors caught during compilation

Java – The Most Popular



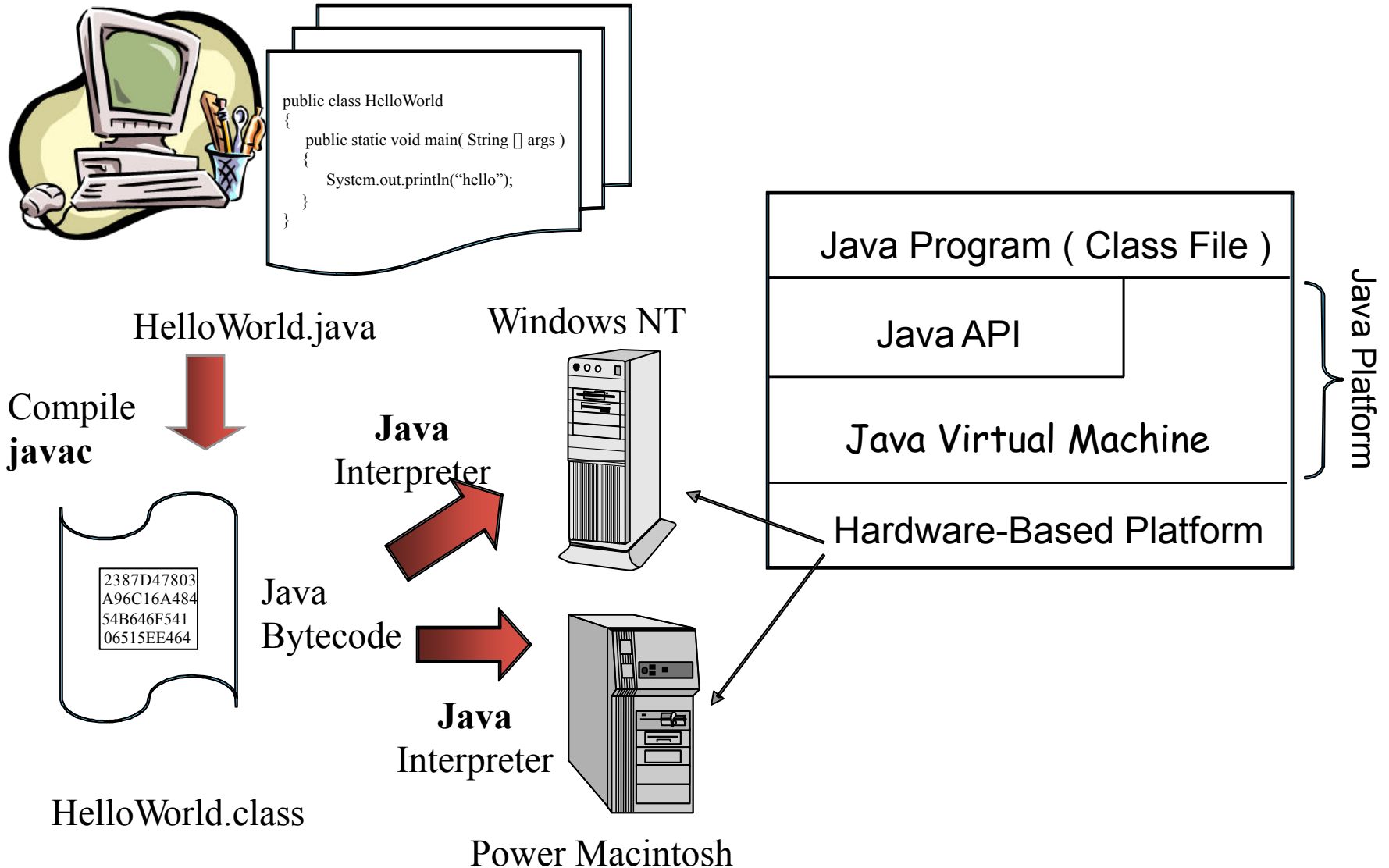
Java – The Most Controversial



Java Editions

- Java 2 Platform, Standard Edition (J2SE)
 - Used for developing Desktop based application and networking applications
- Java 2 Platform, Enterprise Edition (J2EE)
 - Used for developing large-scale, distributed networking applications and Web-based applications
- Java 2 Platform, Micro Edition (J2ME)
 - Used for developing applications for small memory-constrained devices, such as cell phones, pagers and PDAs

Java platform



Java Development Environment

- Edit
 - Create/edit the source code
- Compile
 - Compile the source code
- Load
 - Load the compiled code
- Verify
 - Check against security restrictions
- Execute
 - Execute the compiled

Phase 1: Creating a Program

- Any text editor or Java IDE (Integrated Development Environment) can be used to develop Java programs
- Java source-code file names must end with the ***.java*** extension
- Some popular Java IDEs are
 - NetBeans
 - Eclipse
 - JCreator
 - **IntelliJ**

Phase 2: Compiling a Java Program

- ***javac Welcome.java***
 - Searches the file in the current directory
 - Compiles the source file
 - Transforms the Java source code into bytecodes
 - Places the bytecodes in a file named **Welcome.class**

Bytecodes

- They are not machine language binary code
- They are independent of any particular microprocessor or hardware platform
- They are platform-independent instructions
- Another entity (interpreter) is required to convert the bytecodes into machine codes that the underlying microprocessor understands
- This is the job of the **JVM** (Java Virtual Machine)

JVM (Java Virtual Machine)

- It is a part of the JDK and the foundation of the Java platform
- It can be installed separately or with JDK
- A virtual machine (VM) is a software application that simulates a computer, but hides the underlying operating system and hardware from the programs that interact with the VM
- It is the JVM that makes Java a portable language

JVM (Java Virtual Machine)

- The same bytecodes can be executed on any platform containing a compatible JVM
- The JVM is invoked by the java command
 - *java Welcome*
- It searches the class Welcome in the current directory and executes the main method of class Welcome
- It issues an error if it cannot find the class Welcome or if class Welcome does not contain a method called main with proper signature

Phase 3: Loading a Program

- One of the components of the JVM is the class loader
- The class loader takes the .class files containing the programs bytecodes and transfers them to RAM
- The class loader also loads any of the .class files provided by Java that our program uses

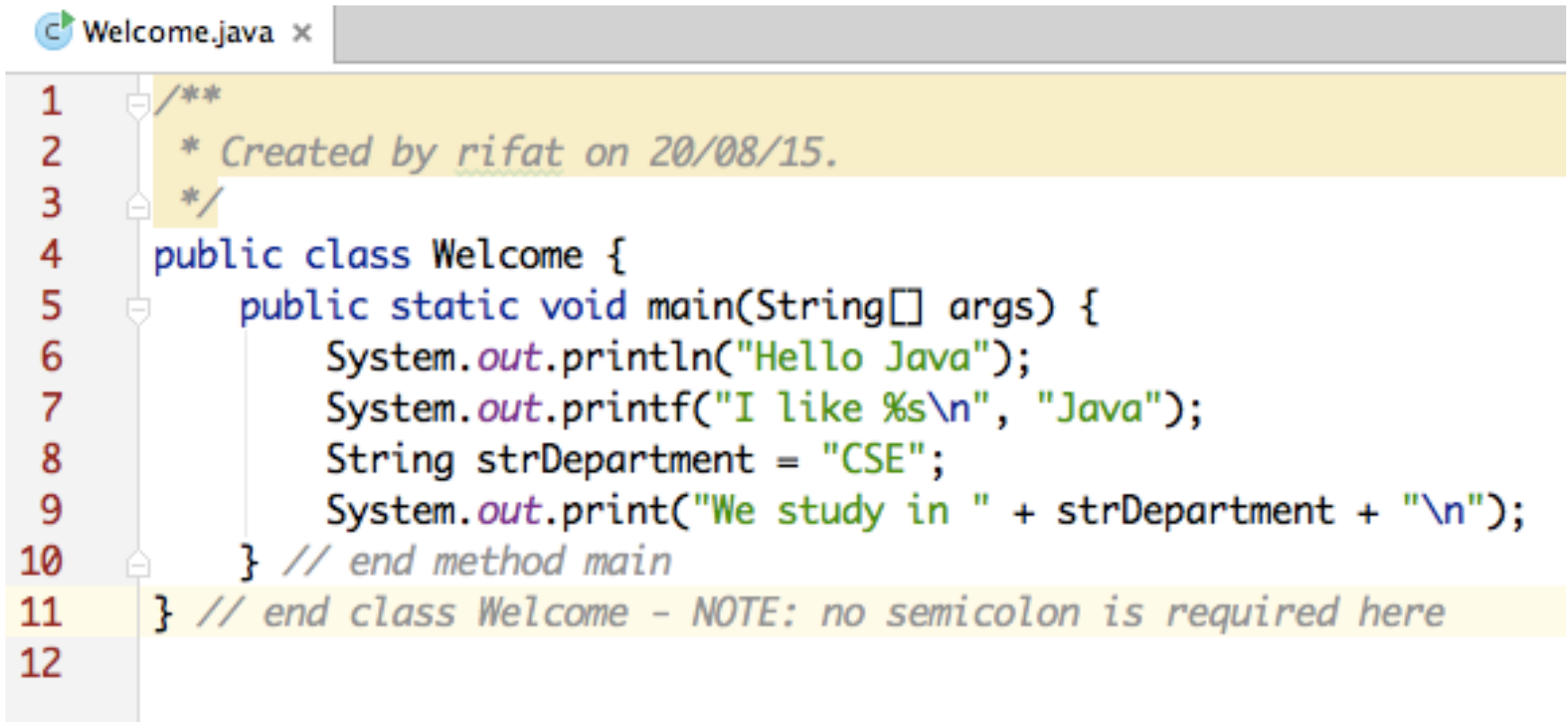
Phase 4: Bytecode Verification

- Another component of the JVM is the bytecode verifier
- Its job is to ensure that bytecodes are valid and do not violate Java's security restrictions
- This feature helps to prevent Java programs arriving over the network from damaging our system

Phase 5: Execution

- Now the actual execution of the program begins
- Bytecodes are converted to machine language suitable for the underlying OS and hardware
- Java programs actually go through two compilation phases
 - Source code -> Bytecodes
 - Bytecodes -> Machine language

Editing a Java Program



The screenshot shows a code editor window titled "Welcome.java". The code is as follows:

```
1  /**
2   * Created by rifat on 20/08/15.
3   */
4  public class Welcome {
5      public static void main(String[] args) {
6          System.out.println("Hello Java");
7          System.out.printf("I like %s\n", "Java");
8          String strDepartment = "CSE";
9          System.out.print("We study in " + strDepartment + "\n");
10     } // end method main
11 } // end class Welcome - NOTE: no semicolon is required here
12
```

Examining Welcome.java

- A Java source file can contain multiple classes, but only one class can be a public class
- Typically Java classes are grouped into packages (similar to namespaces in C++)
- A public class is accessible across packages
- The source file name must match the name of the public class defined in the file with the .java extension

Examining Welcome.java

- In Java, there is no provision to declare a class, and then define the member functions outside the class
- Body of every member function of a class (called method in Java) must be written when the method is declared
- Java methods can be written in any order in the source file
- A method defined earlier in the source file can call a method defined later

Examining Welcome.java

- ***public static void main(String[] args)***
 - **main** is the starting point of every Java application
 - **public** is used to make the method accessible by all
 - **static** is used to make main a static method of class Welcome. Static methods can be called without using any object; just using the class name. JVM call main using the **ClassName.methodName** notation
 - **void** means main does not return anything
 - **String args[]** represents an array of String objects that holds the command line arguments passed to the application. *Where is the length of args array?*

Examining Welcome.java

- Think of JVM as a outside Java entity who tries to access the main method of class Welcome
 - main must be declared as a public member of class Welcome
- JVM wants to access main without creating an object of class Welcome
 - main must be declared as static
- JVM wants to pass an array of String objects containing the command line arguments
 - main must take an array of String as parameter

Examining Welcome.java

- ***System.out.println()***
 - Used to print a line of text followed by a new line
 - **System** is a class inside the Java API
 - **out** is a public static member of class System
 - **out** is an object of another class of the Java API
 - **out** represents the standard output (similar to stdout or cout)
 - **println** is a public method of the class of which out is an object

Examining Welcome.java

- **System.out.print()** is similar to **System.out.println()**, but does not print a new line automatically
- **System.out.printf()** is used to print formatted output like `printf()` in C
- In Java, characters enclosed by double quotes (" ") represents a String object, where String is a class of the Java API
- We can use the plus operator (+) to concatenate multiple String objects and create a new String object

Compiling a Java Program

- Place the .java file in the bin directory of your Java installation
 - ***C:\Program Files\Java\jdk1.8.0_51\bin***
- Open a command prompt window and go to the bin directory
- Execute the following command
 - ***javac Welcome.java***
- If the source code is ok, then javac (the Java compiler) will produce a file called Welcome.class in the current directory

Compiling a Java Program

- If the source file contains multiple classes then javac will produce separate .class files for each class
- Every compiled class in Java will have their own .class file
- .class files contain the bytecodes of each class
- So, a .class file in Java contains the bytecodes of a single class only

Executing a Java Program

- After successful compilation execute the following command
 - ***java Welcome***
 - *Note that we have omitted the .class extension here*
- The JVM will look for the class file *Welcome.class* and search for a *public static void main(String args[])* method inside the class
- If the JVM finds the above two, it will execute the body of the main method, otherwise it will generate an error and will exit immediately

Another Java Program

```
A.java x
1  /**
2   * Created by rifat on 21/08/15.
3   */
4  public class A {
5      private int a;
6
7      public A()
8      {
9          this.a = 0;
10     }
11
12     public void setA(int a)
13     {
14         this.a = a;
15     }
16
17     public int getA()
18     {
19         return this.a;
20     }
21
22
23     public static void main(String args[])
24     {
25         A ob;
26         ob=new A();
27         ob.setA(10);
28         System.out.println(ob.getA());
29     }
30 }
```

Examining A.java

- The variable of a class type is called a reference
 - *ob* is a reference to A object
- Declaring a class reference is not enough, we have to use `new` to create an object
- Every Java object has to be instantiated using keyword **new**
- We access a public member of a class using the dot operator (`.`)
 - Dot (`.`) is the only member access operator in Java.
 - Java does not have `::`, `->`, `&` and `*`



Primitive (built-in) Data types

- Integers
 - byte 8-bit integer (new)
 - short 16-bit integer
 - int 32-bit signed integer
 - long 64-bit signed integer
- Real Numbers
 - float 32-bit floating-point number
 - double 64-bit floating-point number
- Other types
 - char 16-bit, Unicode 2.1 character
 - boolean true or false, *false is not 0 in Java*

Boolean Type

```
Boolean.java x
1  /**
2   * Created by rifat on 21/08/15.
3   */
4  public class Boolean {
5      public static void main(String[] args) {
6          int a = 10;
7          if (a > 0) // if (a) will give compilation error
8          {
9              System.out.println("Inside If");
10         }
11         boolean b = false;
12         if (b)
13         {
14             System.out.println("Inside If");
15         }
16         else
17         {
18             System.out.println("Inside Else");
19         }
20     }
21 }
22
```


Non-primitive Data types

- The non-primitive data types in java are
 - Objects
 - Array
- Non-primitive types are also called reference types

```
1 public class Box {  
2     int L, W, H;  
3  
4     Box(int l, int w, int h)  
5     {  
6         L = l;  
7         W = w;  
8         H = h;  
9     }  
10  
11     public static void main(String[] args) {  
12         Box p; // p is a reference pointing to null  
13         p = new Box(1,2,3); // now the actual object is created  
14     }  
15 }  
16
```

Primitive vs. Non-primitive type

- Primitive types are handled by value – the actual primitive values are stored in variable and passed to methods

int x = 10;

public MyPrimitive(int x) { }

- Non-primitive data types (objects and arrays) are handled by reference – the reference is stored in variable and passed to methods

Box b = new Box(1,2,3);

public MyNonPrimitive(Box x) { }

Java References

- Java references are used to point to Java objects created by new
- Java objects are **always** passed **by reference** to other functions, *never by value*
- Java references act as pointers but does not allow pointer arithmetic
- We cannot read the value of a reference and hence cannot find the address of a Java object
- We cannot take the address of a Java reference

Java References

- We can make a Java reference point to a new object
 - By copying one reference to another

ClassName ref2 = ref1; // Here ref1 is declared earlier

- By creating a new object and assign it to the reference

ClassName ref1 = new ClassName();

- We cannot place arbitrary values to a reference except the special value **null** which means that the reference is pointing to nothing

ClassName ref1 = 100; // compiler error

ClassName ref2 = null; // no problem

Java References

```
Box.java x
1  public class Box {
2      int L, W, H;
3
4      Box(int l, int w, int h)
5      {
6          L = l;
7          W = w;
8          H = h;
9      }
10
11     public static void main(String[] args)
12     {
13         Box b1; // b1 refers to null
14         Box b2; // b2 refers to null
15         b1 = new Box(8, 5, 7); // b1 refers to new object (8, 5, 7)
16         b2 = b1; // b2 refers to b1, so both refers (8, 5, 7)
17         b1 = new Box(3, 9, 2); // b1 refers to new object (3, 9, 2)
18         b1 = b2; // b1 refers to b2, what happens to object (3, 9, 2)
19     }
20 }
21
```

Textbook

- We will follow Java 8
- Java: The Complete Reference, 9th Edition by Herbert Schildt
- Web:
 - <http://teacher.buet.ac.bd/rifat/CSE201.html>
 - <http://cse.buet.ac.bd/moodle/course/view.php?id=88>