Outline

- Java Overview
- Java Examples
- C++ vs java
- Q&A
Java Overview

- Object-Oriented Programming Language (OOPL) by Sun in 1991
  - Programming with One or More Classes
  - Simple Structure
    - w/o header files, preprocessor, struct, operator overloading, multiple inheritance, pointers, etc.
  - Garbage Collection
    - No need to delete or return any storage
  - Dynamic Loading
    - Classes being loaded as needed
  - Platform Independence
    - Java Virtual Machine (JVM)
  - Multithreading
    - Support for multiple threads of execution
Some Differences with C/C++

- Automatic Memory Management
  - Garbage Collector
  - No Dangling Pointers or Memory Leaks
- No Pointer Handling
  - No Explicit Reference/Dereference Operations
- No Makefiles
- No Header Files
  - cf, imported Packages
- No Function Declaration (Similar to C)
- No Default Function Argument
Java Platform

• S/W Platform for Running Java - on Top of any Platforms
  – Java Virtual Machine (JVM)
  – Java Application Programming Interface (Java API)

Collection of ready-made software components - grouped into Packages of classes and interfaces
Java Interpreter

• Implementation of the JVM
  – Executing Java Bytecodes
    • Java bytecodes can be considered as intermediate code instructions for the JVM
    • Java programs, once compiled into bytecodes, can be run on any JVM
How a Java Program Runs

• Compilation and Interpretation
  – Compiler First Translates a Java Program into Java Bytecodes
  • Once
  – Interpreter Parses and Runs Each Java Bytecode Instruction
  • Multiple times on different platforms
Java Program

- Saved in Files, Each of Which Has the Same Name as the public Class
  - Containing Only One public Class
  - Containing Other Non-public Classes

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

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>$ javac HelloWorld.java</code></td>
<td>compile (create HelloWorld.class; bytecode)</td>
</tr>
<tr>
<td><code>$ java HelloWorld</code></td>
<td>start the JVM and run the main method</td>
</tr>
</tbody>
</table>

Hello, World
Memory Layout of a Java Program

Bytecode of Method
Variables in Class
Parameter Variable
Local Variable
Class Object
Array Object
String Object

Method Area
Stack
Heap

Space for **objects**
created by **new** operator

```java
public class MemoryModelTest {
    static int x=0;
    public static void main(String args[]) {
        int a=10, b=20, c;
        c = add(a, b);
    }
    static int add(int a, int b) {
        return(a + b);
    }
}
```

Sample Program:
MemoryModelTest.java
Class

- **Unit of Programming**
  - Java Program: a Collection of Classes
    - Source code in .java files
- **Description (Blueprint) of Objects (Instances)**
  - Common Characteristics
- **Instances Have These Characteristics**
  - Attributes (Data Fields) for Each Object
  - Methods (Operations) That Work on the Objects
Member Access Control

• Way to Control Access to a Class’ Members from Other Classes
  – **private**
    • Accessible only in the class itself
  – Default (package or friendly)
    • Accessible in the same-package subclasses of the class or in the classes of the same package
  – **protected**
    • Accessible in the subclasses of the class or in the classes of the same package
  – **public**
    • Accessible everywhere
Object

- Instance of a Class
- Uniquely Identifiable Entity
  - w/ Its State, Behavior, and Interface
  - Maintaining Data Values in Its Attributes
  - Referenced by a Reference Variable (of Reference Type)
- Inheriting from the Class Object
  - w/ a number of methods
  - toString(), equals(), … & clone()
Managing Objects

• Referencing Objects of Specified Types
  – Objects Created by the new Operator

• Creating Objects by Executing the Constructors
  – Constructor (Function) Overloading

```java
String greeting = new String("hello");
```

• Deleting Objects via Garbage Collection
  – Reference Count for Each Object

Cleanup occurs at the convenience of the Java runtime environment
Java Example: Abstraction

- Online Retailer Such as Amazon.Com
  - Item: Type, Title, Maker, Price, Availability, etc.

```java
class Item { // Class definition
    public String title; // String is a predefined class
    public double price; // double is a primitive data type
    public double SalePrice() { return (price * 0.9); }
}
Item A = new Item(); // Class object definition and creation
// OKAY : A.title, A.price, and A.SalePrice()
```

Attribute of the class
Method of the class
Variable of reference type
Java Example: Encapsulation

• Online Retailer Example Cont’d

```java
class Item {
    public String title;
    public double price;
    private int inStockQuantity;
    public double SalePrice(){ return (price * 0.9);}
    public boolean isAvailable(){
        if(inStockQuantity > 0) return true;
        else return false;
    }
}

Item A = new Item(); // Class object definition and creation

// NOT OKAY: A.inStockQuantity
// OKAY: A.isAvailable()
```

inStockQuantity attribute is not accessible outside of the Item class
Java Example: Inheritance

Online Retailer Example Cont’d

class MusicCDItem extends Item {
    public String singer_name;
}

// Class object definition and creation
MusicCDItem B = new MusicCDItem;

// OKAY: B.singer_name, B.title, B.price, B.SalePrice(),
// and B.isAvailable()
// NOT OKAY: B.inStockQuantity
Java Example: Polymorphism

- Online Retailer Example Cont’d

```java
class Item {
    public String title;
    public double price;
    private int inStockQuantity;
    public double SalePrice(){ return (price * 0.9);}
    public boolean isAvailable(){
        if(inStockQuantity > 0) return true;
        else return false;
    }
    public void specificInfo() {
        System.out.println("no info: a base-class object");
    }
}
```
class MusicCDItem extends Item {
    public String singer_name;
    public void specificInfo(){
        System.out.println("signer name=" + singer_name + ": a derived-class object");
    }
}

public class OnlineRetailer {
    static void printSpecificInfo(Item Item){item.specificInfo();}
    public static void main(String args[]){ ... }
}

Item A = new Item();
MusicCDItem B = new MusicCDItem();

printSpecificInfo(A); // Call Item.specificInfo()
printSpecificInfo(B); // Call MusicCDItem.specificInfo()
// - Another derived class (e.g., MovieDVDItem) with specificInfo()
Static Modifier

- Use: Static Attributes & Static Methods
- Features
  - All Classes Share Static Members
  - It Is Possible to Invoke Static Methods w/o Instantiation
  - In Static Methods, It Is Allowed to Access Non-Static Data or Non-Static Methods of Classes after the Instantiation of the Objects

```java
class A{
    private int i = 5;
    public static printI(){
        System.out.println(i); // error!
        System.out.println(new A().i);
    }
}
```
• Differences between C++ and Java
  – Static Method Invocation
    • C++ : Class::method();
    • Java : Class.method();
  – Static Data Member Initialization
    • C++ : No In-Class Initialization (ANSI/ISO)
    • Java : In-Class Initialization

class A{
public:
    static int i; // declare
    ...
}
int A::i = 0; // define & initialize
Locating Classes

• Filesystem Names Consist of:
  – CLASSPATH
    • Environment Variable Set to a List of Pathnames:
      – Separated by “;” in autoexec.bat on Windows
      – Separated by “:” in a Shell Initialization File on Unix/Linux
        » Bash: $ export CLASSPATH=/a:/a/Java/:.
  – Package Name
    • Name of a Collection of Individual .class Files in a Directory
  – Class Name
Locating Classes Cont’d

- CLASSPATH Tells the Class Loader Where to Begin Looking for All Possible Starting Places
  - Take the Full Name Including the Package Name, e.g., Java.d1.j11
  - Replace the Dots with “/” or “\” and Suffix with “.class,” e.g., Java/d1/j11.class
  - Concatenate It onto Each Element of the CLASSPATH

/a/Java/d1/j11.class
/a/Java/Java/d1/j11.class
./Java/d1/j11.class
Locating Classes Cont’d

• Package Statement (at the Top of Each Source File)
  – Which Package the Class Belongs to
    ```
    package packagename;
    E.g., package d1; (with /a/Java as an element of CLASSPATH)
    ```

• Import Statement
  – Permitting Using a Class Name Directly
    ```
    import packagename.classname;
    E.g., import d1.j11; (with /a/Java as an element of CLASSPATH)
    ```
Example: Locating Classes

- CLASSPATH=/a:/a/Java:
- Current Directory: /a/Java/d1
- File j11.java

```java
// package d1;
public class j11 {
    protected static int i = 1;
}
```

- File j12.java

```java
// package d1;
// import d1.j11;
public class j12 extends j11 {
    public static void main(String args[]) {
        System.out.println("i = " + i);
    }
}
```
Example: Locating Classes Cont’d

- CLASSPATH=/a:/a/Java:
- /a/Java/d1/j11.java

```java
// package d1;
public class j11 {
    protected static int i = 1;
}
```

- /a/Java/d2/j15.java

```java
// package d2;
// import d1.j11;
public class j15 extends j11 {
    public static void main(String args[]) {
        System.out.println("i = " + i);
    }
}
```

martini: java d2/j15.java  // w/ the package statements