COS30041 Creating Secure and Scalable Software

Lecture 01A Java – a crush course

Learning Objectives

- After studying the lecture material, you will be able to
  - Write simple programs in Java

Pre-requisites

- Some experiences in a programming language
- Object-oriented concepts

Introduction to Java

- Java is a programming language like C++ and C#
- It is compiled but not directly into the computer’s machine language
- Java is compiled into an intermediate language called bytecode
- The bytecode can not directly run on a computer, it needs a converter.
- The converter that converts java bytecode into machine code is called a JVM or Java virtual machine

JVM

- For each platform a different JVM is required. Mac computers need a different JVM, Linux machines need a different JVM and Windows machines have their own JVM. The same bytecode provided to these different jvms is converted to respective machine code and generates same results.

Java – more features

- Java is free
- Network programming in Java is easy compared to C++
- Java is object Oriented
- Java has a big collection of pre-written libraries. The documentation for these libraries can be downloaded or viewed online at java.sun.com

Java JDK

- Java development kit or JDK comes with java compiler, libraries and other tools
- The java compiler compiles a source code into bytecode
- The javac command is used to compile a source code into bytecode
- In order to run the bytecode a jvm is needed. The JVM is activated by calling java followed by bytecode file

Java runtime

- A computer may have java runtime without having JDK
- A java runtime has JVM and libraries but no compiler
- In order to check if Java runtime is installed you type java –version at command prompt
- In order to check if java compiler is installed, you may need to check the JDK directories
- Java compiler is usually installed at a location similar to C:\Program Files\Java\jdk1.8.0_191\bin

A simple Java Program

```
import java.util.Scanner;


public class Hello {
        public static final String HELLO = “Hello, ”;
        public static void main(String [] args) {
                Scanner in = new Scanner(System.in);
                String name = in.nextLine();
                System.out.println(HELLO + name + “!”);
        }
}
```

Java Libraries

- The classes in java libraries are grouped in packages
- The packages provide a mechanism of namespace
- These are like directories that group related classes
- Two classes with the name alpha may exist in more than one package but because of being identified with package names can be uniquely identified as package1.Alpha and package2.Alpha

Importing packages

- To use class libraries we use import statement or
- Use fully qualified name of the class
- For instance, to use the Scanner class we can

```
        import java.util.Scanner;
    Or: import java.util.*;
```

- import java.util.*; which imports all classes in the package java.util. Scanner class is in java.util package or
- import java.util.Scanner which imports Scanner class only
- Or use fully qualified name java.util.Scanner in program

Simple input

- System.out represents the display on the monitor
- System.in represents the input from keyboard
- Values could be provided to a java program by linking System.in to an object of Scanner class
- Scanner s = new Scanner (System.in);

Simple input

```
import java.util.Scanner;

public class InputProg {
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);

               //input a string
                String n = sc.nextLine();
                System.out.println("hi "+n);

              //input an integer
                int i = sc.nextInt();
                System.out.println(i);
               }
}
```

Simple input

- The scanner class uses next, nextLine, nextInt, nextFloat and other functions for input of different types
- next, nextLine function is used for an input of type String
- nextInt is used for Integers
- nextFloat is used for floats

Java classes API:

https://docs.oracle.com/javase/8/docs/api/

Simple input

```
import java.util.Scanner;

public class InputProg2 {
        public static void main(String[] args) {
                Scanner s = new Scanner(System.in);
                String st = in.nextLine();
                String[] words = st.split(",");
                System.out.println("First is: " + words[0]);
                System.out.println("Second is: " + words[1]);
                System.out.println(“Third is: " + words[2]);
        }
}
```

Input:

```
  Peter Smith,12 Alice St,0398983333
```

Output:

```
  First is: Peter Smith
  Second is: 12 Alice St
  Third is: 0398983333
```

Java – Data Types

- Primitive data types
  - int
  - float
  - double
  - char
- Objects
  - String
  - …
- arrays – suffix with “[]” (no space between “[” and “]”)

Java – Basic Branching

- **If-then-else** statement

```
if (mark >= 80) {
    grade = “HD”;
} else if (mark >=70) {
    …
} else if (mark >=60) {
  …
} else if (mark >= 50) {
    …
} else {
    …
}
```

- **switch** statement

```
switch (gender) {
  case ‘M’ : …; break;
  case ‘F’ : …; break;
  default : …; break;
}
```

Java – Basic Iteration

- **for** loop

```
for (int i = 0; i < n; i++) {
   System.out.println(“Hi!”);
}
for (String word : list) {
    System.out.print(word + “ “);
}
```

- **while** loop

```
int i = 0;
while (i < n) {
    System.out.println(Hi!”);
}
```

Declaring a Java Class and using Objects

```
public class Point {
    private int x;
    private int y;

    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }
}
```

```
import java.util.Math;

public class LineSeg {
    private Point begin;
    private Point end;

    public LineSeg(Point begin, Point end) {
        this.begin = begin;
        this.end = end;
    }

    public double getDistance() {
        int x1 = begin.getX();
        int y1 = begin.getY();
        int x2 = end.getX();
        int y2 = end.getY();
        double dist = Math.sqrt(
                (x2 – x1)^2 + (y2 – y1)^2 );
        return dist;
    }
}
```

Tying it altogether

```
// assuming Point.java, LineSeg.java and LineProgram.java are in the same folder

public class LineProgram {
    public static void main(String [] args) {
        Scanner in = new Scanner(System.in);
        int x = in.nextInt();
        int y = in.nextInt();
        Point pt1 = new Point(x, y);
        x = in.nextInt();
        y = in.nextInt();
        Point pt2 = new Point(x, y);
        LineSeg line = new LineSeg(pt1, pt2);
        System.out.println("The distance is " + line.getDistance());
    }
}
```

Java packages

- Using packages to organize related Java classes
- Examples:
  - package ed.slsb
  - package ed.dto

Single Inheritance

- Java allows inheritance from a single class only.
- This means that a class can not inherit from two or more classes at the same time.

“Is a” relationship

- Recall that inheritance means “is a” relationship
- Eagle “is a” Bird
- Soft Drink “is a” Food
- Dog “is an” animal

“Can do” relationship through Interfaces

- Sometimes a class has a functionality of some other class which can not be described by the “is a” relationship
- A Student can have a functionality of the Singer class when the Student objects **<u>can</u>** sing like Singer objects but Student class may not be a sub class of Singer class
- The “can do” relationship is established using the **Interfaces**

Interfaces - 1

- **Interfaces** are used to describe if objects of a class “can do” something mentioned in the Interface which means that:
- **Interfaces** allow us to identify if an object of any class has a certain function in it or not
- If a class says that it <u>implements</u> an **Interface** then that class would definitely have functions described in that **interface**

Interfaces - 2

- They have only empty declarations of functions in them and don’t have a defined function
- The bodies of functions are kept empty so that classes that implement these interfaces could define function bodies as per their own needs

implements keyword

- When a class inherits from another class it uses the keyword **extends**
- When a class claims that its objects can do functions described in an Interface, it uses the keyword **implements**
- A class can extend only one class but can implement many interfaces
- A class can use extends and implements keywords together

Interfaces - Example

```
public interface Singer {
public void sing ();
}


public interface Dancer {
    public void dance();
}
```

Interfaces have one or more function declarations but no definition of functions. We put ; right after their signatures.

Interface Singer is saved as Singer.java and interface Dancer is saved as Dancer.java

Both are compiled before use

```java
public class Student implements Singer, Dancer {
    String name;
    int age;
    int student_id;
    public Student(String nm, int ag, int st_id) {
            name = nm;
            age = ag;
            student_id = st_id;
    }
    public void study() {
            System.out.println("I am studying");
    }
    public void sing() {
            System.out.println("la la la I am singing");
    }
    public void dance () {
            System.out.println("I am dancing");
    }
}
```

```java
public class Stdriver {

public static void main(String[ ] args) {
Student s1 = new Student("John",23,67548);
        s1.study();
        s1.sing();
        s1.dance();

  }

}
```

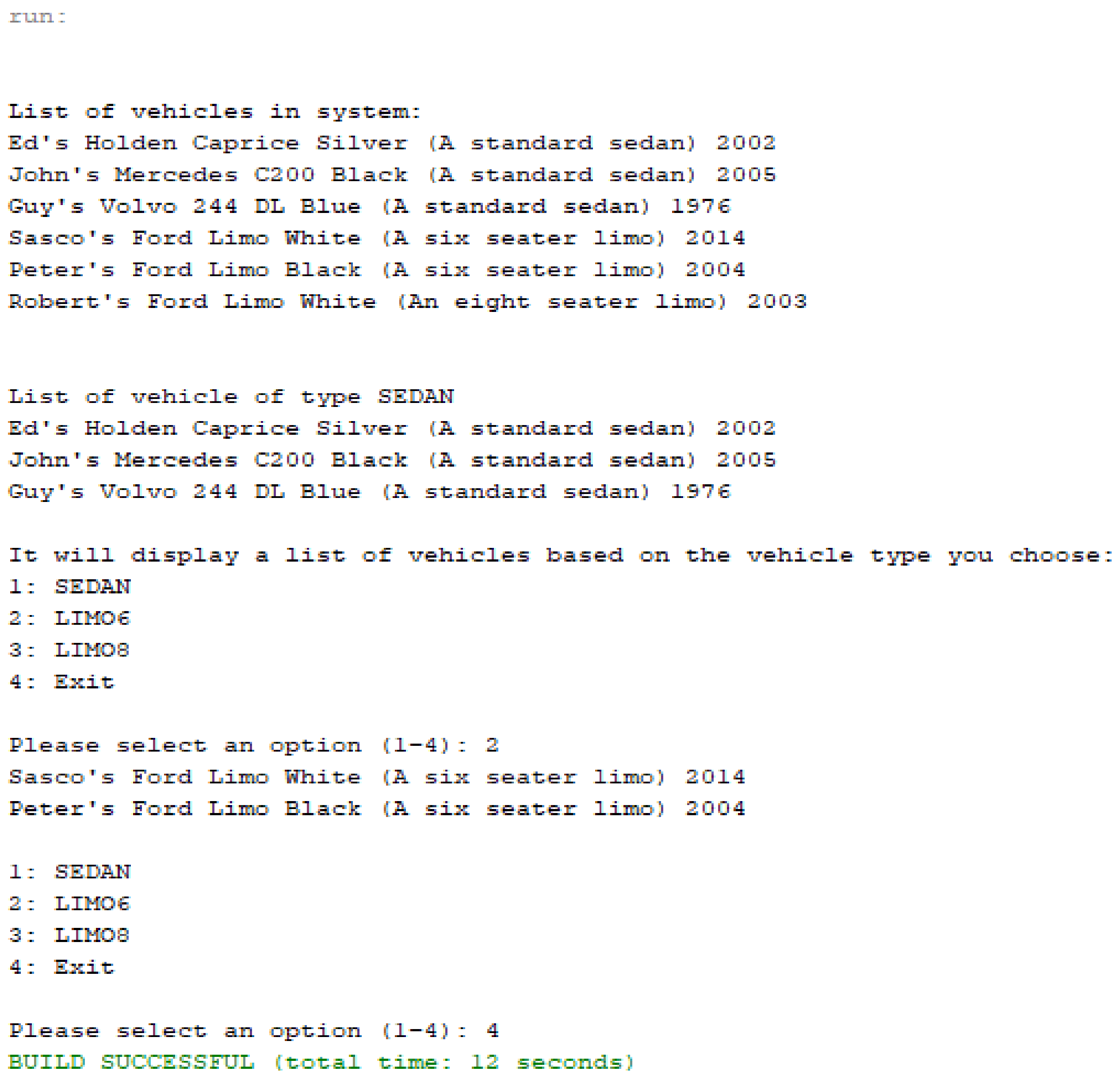
Java Programming

No portfolio task required to be submitted in week 1.

After completing Lab_01a_Setup_NB & GF, work on

- Lab_01b_Using_NetBeans for Java programming
- Practice task:

  Modify the code “VehicleHireApp.java” by adding a user menu for choosing a vehicle type. After a vehicle type is chosen, a list of vehicles for this type is displayed. A sample run is show below:



```
run:


List of vehicles in system:
Ed's Holden Caprice Silver (A standard sedan) 2002
John's Mercedes C200 Black (A standard sedan) 2005
Guy's Volvo 244 DL Blue (A standard sedan) 1976
Sasco's Ford Limo White (A six seater limo) 2014
Peter's Ford Limo Black (A six seater limo) 2004
Robert's Ford Limo White (An eight seater limo) 2003


List of vehicle of type SEDAN
Ed's Holden Caprice Silver (A standard sedan) 2002
John's Mercedes C200 Black (A standard sedan) 2005
Guy's Volvo 244 DL Blue (A standard sedan) 1976

It will display a list of vehicles based on the vehicle type you choose:
1: SEDAN
2: LIMO6
3: LIMO8
4: Exit

Please select an option (1-4): 2
Sasco's Ford Limo White (A six seater limo) 2014
Peter's Ford Limo Black (A six seater limo) 2004

1: SEDAN
2: LIMO6
3: LIMO8
4: Exit

Please select an option (1-4): 4
BUILD SUCCESSFUL (total time: 12 seconds)
```

Menu

```
Scanner sc = new Scanner(System.in);
  int option = 0;

 System.out.println("It will display a list of vehicles based on the vehicle type you choose:");
  while (option != 4) {
    System.out.println("1: SEDAN");
    System.out.println("2: LIMO6");
    System.out.println("3: LIMO8");
    System.out.println("4: Exit");
    System.out.print("\nPlease select an option (1-4): ");
    option = sc.nextInt();
    sc.nextLine(); //skip '\n'

    // switch statement that invokes the various methods of the class based on the option
    // selected by the user
    switch (option) {
      case 1:
        ……………..
        ………………

 } //end while
```

Display a menu