300130

Internet Programming

Lecture 1

INTRODUCTION

1

WESTERN SYDNEY UNIVERSITY

Main Topics of Lecture

- Unit Information
- Why Java
- Java Basics

Online Teaching

- Zoom Meeting
  - Lecture information published on vUWS
  - Tutorial information published on vUWS
    - All submissions on vUWS
    - Demonstration and personal assistance etc via Breakout Room

Assessment

• 3 Random Workshops	15%, 5% each
• 1 Individual Assignment	15%
• 1 Group Assignment	25%
• Final Exam	45%

  - 2 hours
  - open book

Main objectives

- Master principles & techniques of OOP
  - Java Programming
- Master Internet
  - Web pages with forms and CGI
  - GUI
  - Java networking
  - JDBC
  - Java Server Pages etc

History of Java

- Originally for intelligent consumer-electronic devices
    - Sun Microsystems funded a project Green in 1991
    - resulted in a C++ based OOP language called java
    - write once, run anywhere
- Useful for creating web pages with dynamic content
    - World Wide Web exploded in 1993
    - Java was formally introduced in 1995
- Now used to:
    - Develop large-scale enterprise applications
    - Enhance web server functionality
    - Provide applications for consumer devices (cell phones, small phones, etc.)
    - Develop robotics software

Why Java?

- Prominent in OOP
- Portable at **bytecode** level
- Similar syntax to C/C++
- Other keywords
  - Interpreted, robust, secure, multithreaded, distributed
- Main executables
  - javac, java, appletviewer, javadoc

Java Class Libraries

- Java programs consist of classes
  - Include methods that perform tasks
- Java provides class libraries
  - Known as Java APIs (Application Programming Interfaces)
- To use Java effectively, you must know
  - Java programming language
  - Extensive class libraries

Typical Java Development Environment

Java programs go through five phases

- Edit
  - Write programs using an editor
  - Store programs with the `.java` file name extension: `test.java`
- Compile
  - Use `javac` (the Java compiler) to create bytecodes from source code program and store them in `.class` files
  - `Javac test.java  -> test.class`
- Load
  - Class loader reads bytecodes from `.class` files into memory

Typical Java Development Environment

- Verify
  - Bytecode verifier examines bytecodes to ensure that they are valid and do not violate security restrictions
- Execute
  - Java Virtual Machine (JVM) translates bytecodes into machine language
  - Use `java` to load, verify and execute the bytecodes in .class files

Integrated Development Environments (IDEs)

- Provide tools that support the software development process, such as editors, debuggers for locating logic errors.
  - Eclipse (www.eclipse.org)
  - Netbeans (www.netbeans.org)
  - Intellij IDEA (www.jetbrains.com)

Simple Java Program

```
/* filename: SimpleJava.java */
public class SimpleJava {
    public static void main(String[] args) {
        System.out.println("Welcome to IP!");
    }
} // end class SimpleJava
```

- Compile: javac SimpleJava.java
- Execute: java SimpleJava

Run it

Command-Line Arguments

```
// filename: SimpleJava1.java
public class SimpleJava1 {
    public static void main(String[] args) {
       for(int i=0;i<args.length;i++)
          System.out.print(args[i] + " ");
    }
}
```

Run it

- **Compile**: javac SimpleJava1.java
- **Execute**: java SimpleJava1 Welcome to IP
- args.length is 3

Common Programming Error

- It is an error for a `public` class to have a file name that is NOT identical to the class name (plus the `.java` extension).
- It is an error to declare more than one `public` class in the same file.

Example

```java
// Fig. 2.7: Addition.java
// Addition program that displays the sum of two numbers.
import java.util.Scanner; // program uses class Scanner

public class Addition
{
   // main method begins execution of Java application
   public static void main( String args[] )
   {
      // create Scanner to obtain input from command window
      Scanner input = new Scanner( System.in );

      int number1; // first number to add
      int number2; // second number to add
      int sum; // sum of number1 and number2

      System.out.print( "Enter first integer: " ); // prompt
      number1 = input.nextInt(); // read first number from user

```

`import` declaration imports class `Scanner` from package `java.util`.

Declare and initialize variable `input`, which is a `Scanner`.

Declare variables `number1`, `number2` and `sum`.

Read an integer from the user and assign it to `number1`.

```java
      System.out.print( "Enter second integer: " ); // prompt
      number2 = input.nextInt(); // read second number from user

      sum = number1 + number2; // add numbers

      System.out.printf( "Sum is %d\n", sum ); // d

   } // end method main

} // end class Addition
```

Read an integer from the user and assign it to `number2`.

Calculate the sum of the variables `number1` and `number2`, assign result to `sum`.

Display the sum using formatted output.

```
Enter first integer: 45
Enter second integer: 72
Sum is 117
```

Two integers entered by the user.

Run it

Class and Object

- What is a class?
  - Blueprint or prototype defining variables and methods common to all objects of a certain kind
- What is an object?
  - A software bundle of variables and related methods
  - An instance of a class
- Every Java program is a class definition.
- Executable Java class must include a **main** function.

Method and Field

- Class contains one or more fields
  - Represent an object's state
  - Specified by Instance variables
  - e.g. a person's name, birth date, address and phone number
- Class provides one or more methods
  - Represent an object's behaviour
  - e.g. deposit or withdraw money from a bank account; calculate a person's age
  - One or more methods manipulate the instance variables

Constructor

- Initialize an object of a class when the object is created
- Java requires a constructor for every class
- Java will provide a default no-argument constructor if none is provided
- Called when keyword **new** is followed by the class name and parentheses
- No destructor is required for Java.
- The format of a constructor:
  - public ClassName(args) {…}

No return type

The same as class name

Instance Variables

- Declared in a class declaration
  - Outside of any methods
- Each object has a separate instance of the variable
- Exist
  - before methods are called on an object
  - while the methods are executing
  - after the methods complete execution

Local Variables

- Declared in the body of method
- Can only be used within that method
- A method’s parameters are local variables of the method

```java
// Fig. 3.5: Account.java
// Account class with a constructor that initializes the name.

public class Account
{
   private String name; // instance variable

   // constructor initializes name with parameter name
   public Account(String name) // constructor name is class name
   {
      this.name = name;
   }

   // method to set the name
   public void setName(String name)
   {
      this.name = name;
   }

   // method to retrieve the name
   public String getName()
   {
      return name;
   }
} // end class Account
```

Fig. 3.5 | Account class with a constructor that initializes the name.

Access Modifiers: **private & public**

- **private** variables and methods are accessible only to methods of the class in which they are declared
- Declaring instance variables **private** is known as data hiding
- Classes often provide **public** methods to allow the class's clients to set or get private instance variables.

Design a Class

- class comment explaining the purpose of the class
- class name and qualifiers
- class body
    - fields
        - instance variables
        - static variables
    - methods
        - constructors
        - instance methods
        - static methods
        - local variables

Class Instance Creation

- Java is extensible
  - Programmers can create new classes
- Class instance creation expression
  - Keyword **new**
  - Then name of class to create and parentheses
- Calling a method
  - Object name, then dot separator (**.**), then method name and parentheses

```java
// Fig. 3.6: AccountTest.java
// Using the Account constructor to initialize the name instance
// variable at the time each Account object is created.

public class AccountTest
{
   public static void main(String[] args)
   {
      // create two Account objects
      Account account1 = new Account("Jane Green");
      Account account2 = new Account("John Blue");

      // display initial value of name for each Account
      System.out.printf("account1 name is: %s%n", account1.getName());
      System.out.printf("account2 name is: %s%n", account2.getName());
   }
} // end class AccountTest
```

```
account1 name is: Jane Green
account2 name is: John Blue
```

Fig. 3.6 | Using the `Account` constructor to initialize the `name` instance variable at the time each `Account` object is created.

Run it

Package

- Packages are used to group classes
- Every class belongs to a package. It is added to a package when it is compiled.
- There are two ways to use classes from another package
  - Use the **import** statement
  - Use the fully qualified name of the class (*package.class*)

Default Package

- Most classes you'll use must be imported explicitly
- Classes in the same package are implicitly imported
- There's a special relationship between classes that are compiled in the same directory
  - By default, such classes are considered to be in the same package—known as the default package.

Java.lang

- By default, package `java.lang` is imported in every Java program
- `java.lang` is the only package in the Java API that does not require an `import` declaration.

API and Packages

Package	**Purpose**	**Sample classes**
Java.lang	Language support	Math
Java.util	Utilities	Random
Java.io	Input and output	PrintStream
Java.awt	Abstract Windowing Toolkit	Color
Java.applet	Applets	Applet
Java.net	Networking	Socket
Java.sql	Database access through SQL	ResultSet

Identifiers

- Class, method and variable names are identifiers.
- By convention all use camel case names.
- Class names begin with an uppercase letter, and method and variable names begin with a lowercase letter.

Primitive and Reference Types

- Types in Java are divided into two categories
  - primitive
  - reference
- Primitive Types

  boolean, char, byte, short, int, long, float, double
- Reference types
  - All other types
  - classes, which specify the types of objects, are reference types.

Type Initialization

- Instance variables are initialized by default
  - Variables of types **byte, char, short, int, long, float** and **double** are initialized to 0.
  - Variables of type **boolean** are initialized to false.
  - Reference-type variables are initialized to the value null.
  - The default value for type **String** is null.
- Local variables are not automatically initialized

Type Conversion

- Type Conversion (type casting):
  - Auto type casting from smaller to larger
    - `byte<short<int<long<float<double`
    - `char<int`
  - Larger to smaller: explicit casting
    - `Type2 type2var = (Type2)type1Var`
- String
  - String ss = "This is the string1" + "This is the string2"
- Arithmetic Operators
  - `+, -,*, /, %, ++, --`

Conditionals

- Logical Operators:
  - `==, !=, >, <, >=, <=, &&, ||, !`
  - *Obj* **instanceof** *ClassName*

- Conditionals:
  - if (booleanExpression)
      {statements;}
    else
      {statements;}
  - switch(integralExpression) {switchBody;}
    - The integralExpression must yield a value of char, byte, short, or int type

Loops

- while (booleanExpression)

      { statements;}

- do

      {statements;}

  while (booleanExpression)

- for (start;end;changing)

      {statements;}

Arrays

- int[] ia = new int[10];
- double[] da = new double[10];
- String[] ss = new String[10];
- Circle[] Circles = new Circle[10];
  - for (int i=1;i<10;i++)

    Circles[i] = new Circle(100, 300, i);
- int[][] ia = new int[10][20];

An Example

```java
// Fig. 7.3: InitArray.java
// Initializing the elements of an array with ar

public class InitArray
{
   public static void main( String args[] )
   {
      // initializer list specifies the value for each element
      int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

      System.out.printf( "%s%8s\n", "Index", "Value" ); // column headings

      // output each array element's value
      for ( int counter = 0; counter < array.length; counter++ )
         System.out.printf( "%5d%8d\n", counter, array[ counter ] );
   } // end main
} // end class InitArray
```

Declare `array` as an array of `int`s

Compiler uses initializer list to allocate array

```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```

Run it

Reading

- **Java How to Program**
  - Chapter 1-7