COMP9024: Data Structures and Algorithms

Week One: Java Programming Language (I)

Hui Wu

Session 1, 2015

http://www.cse.unsw.edu.au/~cs9024

Outline

- Classes and objects
- Methods
- Primitive data types and operators
- Arrays
- Control flow
- I/O streams

Why Java?

- Simple
- Architecture neutral
- Object oriented
- Portable
- Distributed
- High performance
- Multithreaded
- Robust
- Dynamic
- Secure

Java Compiler and Virtual Machine

- Java compiler converts the source program into bytecodes – the machine language of Java Virtual Machine (Java VM).
- Java VM converts the bytecodes into the machine code which can be run directly on the local computer.

Classes

- A class consists of
  - a collection of *variables*, or fields, and
  - a set of functions, called *methods*.
- A class serves as a template of objects.

Naming Conventions

- Java is case-sensitive; max, mAx, and maX are three different names.
- Class names begin with a capital letter.
- All other names begin with a lowercase letter.
- Subsequent words are capitalized: firstValue.
- Underscores are not used in names.

A Counter Class

```
Public class Counter {
   protected int count; // a simple integer instance variable
  /** The default constructor for a Counter object */
   Counter() { count=0; }
   /** three methods are defined as follows */
   public int getCount() { return count; }
   public void incrementCount() { count++; }
   Public void decrementCount() { count--; }
                        }
```

7

Class Modifiers

- Abstract
  - Describes a class that has abstract methods. A abstract method is declared with the abstract keyword and is empty.
- Final
  - Describes a class with no subclasses.
- Public
  - Describes a class that can be instantiated or extended.

8

Variables

- Instance variables (non-static fields)
  - fields declared without the static keyword
- Class variables (static fields)
  - fields declared with the static keyword
- Local variables
  - Variables declared within a method
- Parameters

9

Variables Modifiers (1/2)

- The scope of an instance variable is controlled by the following variable modifiers:
  - public
    - Anyone can access it
  - protected
    - Only the methods of the same package or of its subclasses can access it
  - private
    - Only the methods of the same class can access them
  - If none of the modifiers above are used, then the variable is friendly. Friendly variables can only be accessed by classes in the same package.

10

Variables Modifiers (2/2)

- In addition to scope variable modifiers, there are also usage modifiers:
  - static
    - Declares a variable that is associated with a class, not with individual instances of that class.
  - final
    - Declares a variable that must be assigned an initial value, and then can never be assigned a new value after that.

11

Methods

- Syntax:

modifiers type name($type_0$ $para_0$, … $type_{n-1}$ $para_{n-1}$){
  // method body
  }

- type defines the return type of the method.

12

Method Modifiers (1/2)

- The scope of a method is controlled by the following variable modifiers:
  - public
    - Anyone can call it.
  - protected
    - Only the methods of the same package or of subclasses can call it.
  - private
    - Only the methods of the same class can call it.
  - If none of the modifiers above are used, then the method is friendly. Friendly methods can only be called by objects of classes in the same package.

Method Modifiers (2/2)

- The above modifiers may be followed by additional modifiers:
  - final
    - This method cannot be overridden by a subclass.
  - static
    - This method is associated with the class itself, not with a particular instance of the class.

Constructors

- A class has a special method, called constructor. It is used to initialize the object instantiated from the class.

Overloading Methods

- Methods within a class can have the same name if they have different parameter lists.
- Overloaded methods are differentiated by the number and the type of the arguments passed into the method.

```
public class DataArtist { ...
  public void draw(int i) { ... }
  public void draw(double f) { ... }
  public void draw(int i, double f) { ... } }
```


The Main Method

- The main method is the method through which a java program is executed.
- Syntax:

```
public static void main(String [] args)
 {
   \\ main method body
 }
```


Objects (1/3)

- An object is an instance of a class.
- Created by using the new operator.
- A new object is dynamically allocated in memory and all instance variables are initialized to standard default values:
  - null for objects
  - 0 for all base types
  - false for boolean variable

Objects (2/3)

- The constructor for the new object is called with the parameters specified.
- After the constructor returns, the new operator returns a reference (memory address) to the newly created object.

Objects (3/3)

```
public class Example
 {
   public static void main (String args)
    {
      Counter c;
      Counter d = new Counter();
       c = new Counter();
       d = c;
    }
 }
```


String Objects

- A string is a sequence of characters from the Unicode internal characters set.
- String objects are *immutable*, which means that once created, their values cannot be changed.
- An example of Java strings:

  String s ="kilo" + "meters"

  where String the name of Java String class and + is the concatenation operator.

Object References

- The variables and methods of an object are referenced via the dot operator.
- Examples of object references:

```
c = new Counter();
c.incrementCount();
```


Primitive Types

- boolean true or false
- char 16-bit Unicode character
- byte 8-bit signed two's complement integer
- short
- int
- long
- float
- double

Same as in C

No unsigned numbers!

Enum Types

- Java supports enumerated types, called Enum.
- Syntax:

  modifier enum name {valueName$_0$, valueName$_1$, …, valueName$_{n-1}$}

An Example of Enum Types

```
public class DayTripper {
  public enum Day { MON, TUE, WED, THU, FRI, SAT, SUN};
  public static void main{string[] args) {
    Day d=Day.MON;
    System.out.printl("Initially s is" + d);
    d=DAY.WED;
    Day t=Day.valueOf("Wed)";
    System.out.printl("I say d and t are the same:" + (d==t));
      }
    }
```


Literals

- Object literal: null
- Boolean: true and false
- Integer:
  - Default type is int: -52, 172
  - Long integer ended with L or l: 186L, 54l
- Float point
  - Default type is double: 12.5, –1.25
  - Type float ended with F or f: 12.5F, 1.5f
- Character
- Strings literals: "dogs cannot fly"

Operators (1/3)

Operator	Purpose
+	addition of numbers, concatenation of Strings
+=	add and assign numbers, concatenate and assign Strings
-	subtraction
-=	subtract and assign
*	multiplication
*=	multiply and assign
/	division
/=	divide and assign
%	take remainder
%=	take remainder and assign
++	increment by one
--	decrement by one

Operators (2/3)

Operator	Purpose
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
!	boolean NOT
!=	not equal to
&&	boolean AND
||	boolean OR
==	boolean equals
=	assignment
~	bitwise NOT
?:	conditional

Operators (3/3)

Operator	Purpose
Instanceof	type checking
|	bitwise OR
|=	bitwise OR and assign
^	bitwise XOR
^=	bitwise XOR and assign
&	bitwise AND
&=	bitwise AND and assign
>>	shift bits right with sign extension
>>=	shift bits right with sign extension and assign
<<	shift bits left
<<=	shift bits left and assign
>>>	unsigned bit shift right
>>>=	unsigned bit shift right and assign

Control Statements

- if-else
- switch
- while loop
- for loop
- do-while loop

if-else

```
void applyBrakes(){
   if (isMoving){ // the "if" clause: bicycle must moving currentSpeed--
    ; // the "then" clause: decrease current speed } }

void applyBrakes(){
   if (isMoving) { currentSpeed--; }
   else { System.err.println("The bicycle has already stopped!"); } }
```


switch

```
class SwitchDemo {
  public static void main(String[] args) {
  int day = 1;
  switch (day)
   { case 1: System.out.println("Saturday");
             break;
    case 2: System.out.println("Sunday");
             break;
   default: System.out.println("Workday");
             break;
  }}}
```


while Loop

```
class WhileDemo{
   public static void main(String[] args){
     int count = 1;
     while (count < 11)
      { System.out.println("Count is: " + count);
        count++;
      }
     } }
```


do-while Loop

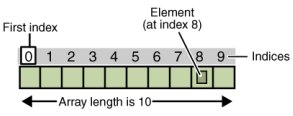
```
class DoWhileDemo{
   public static void main(String[] args){
     int count = 1;
     do {
          System.out.println("Count is: " + count);
          count++;
        } while (count <= 11);
   } }
```


for loop

```
class ForDemo {
   public static void main(String[] args){
     for(int i=1; i<11; i++)
       System.out.println("Count is: " + i);
      }
     }
```


Arrays (1/4)

- An *array* is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created. After creation, its length is fixed.



Arrays (2/4)

```
class ArrayDemo {
    public static void main(String[] args) {
      int [] anArray; // declares an array of integers
      anArray = new int[10]; /** allocates memory for 10 integers */
      for (int i=0; i<anArray.length; i++)
          anArray[i]=i;
       }
      }

   /* anArray.length returns the length of anArray, i.e. 10. */
```


Arrays (3/4)

- The System class has an arraycopy method that can be used to efficiently copy data from one array into another:

```
public static void arraycopy(Object src, int srcPos, Object dest, int
destPos, int length)
```

The two Object arguments specify the array to copy *from* and the array to copy *to*. The three int arguments specify the starting position in the source array, the starting position in the destination array, and the number of array elements to copy.

Arrays (4/4)

```
class ArrayCopyDemo {
  public static void main(String[] args) {
    char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't', 'e', 'd' };
    char[] copyTo = new char[7];
    System.arraycopy(copyFrom, 2, copyTo, 0, 7);
    System.out.println(new String(copyTo));
   }
  }
```


Nested Classes

- A nested class is a member of its enclosing class and, as such, has access to other members of the enclosing class, even if they are declared private.
- A nested class can be declared private, public, protected, or *package private* while an outer class can only be declared public or *package private.*

```
class OuterClass { ...
  class NestedClass { ... }
                }
```


Why Nested Classes?

- It is a way of logically grouping classes that are only used in one place.
- It increases encapsulation.
- Nested classes can lead to more readable and maintainable code.

For more about nested classes, refer to http://java.sun.com/docs/books/tutorial/java/javaOO/innerclasses.html

Packages (1/4)

- **Java package** is a mechanism for organizing Java classes into namespaces.
- A package provides a unique namespace for the types it contains.
- Classes in the same package can access each other's protected members.
- A package can contain the following kinds of types.
  - Classes
  - Interfaces
  - Enumerated types
  - Annotations

Packages (2/4)

- In Java source files, the package that the file belongs to is specified with the package keyword.

```
package java.awt.event;
```

- To use a package inside a Java source file, use an import statement to import the classes from the package.

```
//imports all classes from the java.awt.event package:
  import java.awt.event.*;
//imports only the ActionEvent class from the package:
  import java.awt.event.ActionEvent;
```


Packages (3/4)

- After either of these import statements, the ActionEvent class can be referenced using its simple class name:

```
ActionEvent myEvent = new ActionEvent();
```

- Classes can also be used directly without an import statement by using the fully-qualified name of the class. For example,

```
java.event.ActionEvent myEvent = new java.awt.event.ActionEvent();
```


Packages (4/4)

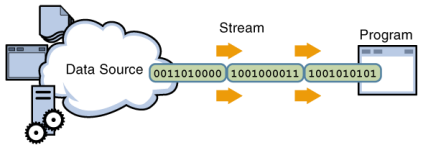
- Packages are usually defined using a hierachical naming pattern, with levels in the hierarchy separated by periods (.) (pronounced "dot").
- In general, a package name begins with the top level domain name of the organization and then the organization's domain and then any subdomains listed in reverse order. The organization can then choose a specific name for their package. Package names should be all lowercase characters whenever possible.
- For more naming details, refer to Java Language Specification(http://java.sun.com/docs/books/jls/third_edition/html/packages.html#7.7).

I/O Streams (1/3)

- An *I/O Stream* represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
- No matter how they work internally, all streams present the same simple model to programs that use them:
  - A stream is a sequence of data.

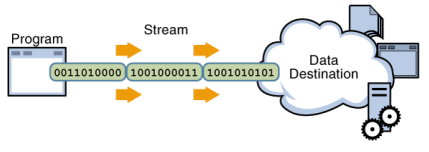
I/O Streams (2/3)

- A program uses an *input stream* to read data from a source, one item at a time:



I/O Streams (3/3)

- A program uses an *output stream* to write data to a destination, one item at time:



Simple Output Methods

- Java provides a built-in static object, called System.out to perform output to the standard output device. This object provides the following methods for buffered output stream:
  - print(Object o): Print the object o using its toString methood.
  - print (String s): Print the string s.
  - print(baseType b): Print the base type value of b.
  - println(String s): Print the string s, followed by the newline. character.

Simple Input Methods (1/4)

- The Scanner class reads the input and divides it into tokens, which are contiguous strings of characters separated by delimiter. The default delimiters are whitespace, tab and newline. So tokens are separated by strings of whitespaces, tabs and newlines.
- The resulting tokens may then be converted into values of different types using the various next methods.

Simple Input Methods (2/4)

- The Scanner class provides the following methods:
  - hasNext(): Return true if only if there is another token in the input stream.
  - Next(): Return the next token string in the input stream; generate an error if there are no more tokens left.
  - nextType(): Return the next token in the input stream, returned as the base type corresponding to Type; generate an error if there are no more token left or if the next token cannot be interrpreted as a base type correspondong to Type.

Simple Input Methods (3/4)

  - hasNextType(): Return true if and only if there is another token in the input string and it can be interpreted as the corresponding base type Type , where Type can be Boolean, Byte, Double, Int, Long, or Short.
- The Scanner class also provides the following methods for processing input line by line:
  - hasNextLine(): Return true if and only if the input stream has another line of text

Simple Input Methods (4/4)

  - nextLine(): Adance the input past the current line and returns the input that was skipped.
  - findInLine(String s): Attempt to find a string matching the (regular expression) pattern s in the current line. If the pattern is found, it is returned and the scanner advances to the first character after this match. If not, the scanner returns null and does not advance.

An Example of Simple I/O (1/2)

```
import java.io.*;
import java.util.Scanner;
public class InputExample {
  public static void main(String args[]) throws IOException {
    Scanner s = new Scanner(System.in);
    System.out.print("Enter your height in centimeters: ");
    float height = s.nextFloat();
    System.out.print("Enter your weight in kilograms: ");
```

An Example of Simple I/O (2/2)

```
    float weight = s.nextFloat();
    float bmi = weight/(height*height)*10000;
    System.out.println("Your body mass index is " + bmi + "."); }
    System.out.print("Please enter an integer: ");
    while (!s.hasNextInt()) {
      s.nextLine();
      System.out.print("That's not an integer; please enter an integer: ");
                              }
    int i = s.nextInt();
}
}
```


Run A Java Program

You need to install **a JAVA IDE (Integrated Development Environment),** either Netbeans (https://netbeans.org/), or Eclipes (https://www.eclipse.org/).

- Step One: Create your source file, say HelloWorldApp.java
- Step Two: Compile your program into bytecodes:

  javac HelloWorldApp.java

  Java compiler will convert the source code into bytecodes and store it in the file HelloWorldApp.class
- Run the bytecodes:

  java HelloWorldApp

References

1. Chapter 1, Data Structures and Algorithms by Goodrich and Tamassia.
2. The Java™ Tutorials (http://java.sun.com/docs/books/tutorial/).