ICT131

Introductory Programming and Object-oriented Concepts Using Java

Objective

The aim of this course is to introduce you to Java programming and the basic concepts in object-oriented programming.

Important Points to Remember

1. Mark Deduction for Late Submissions of Tutor-Marked Assignments (TMA):
   - Unless otherwise advised, the assignment submission due date is one day before the scheduled class day. The deadline time is 2359 hours on the due date.
   - No extension can be given to TMA cutoff dates
2. Successful submission of TMAs:
   - Upon successful submission, you should see a receipt number on the screen. Please take note of this receipt number as proof of your TMA submission.

Important Points to Remember

3. Ensure that the correct file naming convention is adopted for TMAs:
   - Refer to the MyUniSIM Student Guide (pages 13 & 14)
4. Collusion in Assignments (TMA) :
   - A serious academic offence. Turnitin will flag all instances of copying done in assignments.
   - TMA is an individual assignment so it should be a students own work

Important Points to Remember

5. Correspondence with UniSIM using MyMail account:
   - We will only accept correspondences sent from you using your UniSIM MyMail account (xxxx@unisim.edu.sg).
6. Approach Student Relations Department if you have any query :
   - Call 6248 9111
   - or email to students@unisim.edu.sg

ICT131 - Assessment

Assessment	Description	Weight Allocation
Assignment 1	On-line Quiz	9%
Assignment 2	TMA	21%
Examination	**Close Book**	70%
	TOTAL	**100%**

- To be sure of a pass result, you need to achieve scores of 40% in each component.
- TMA – 12 hours grace period. Thereafter 10 marks per day.

6

ICT 131 – Seminar Sessions

- Total 6 lessons
  - 3 hrs per session
  - practical sessions
- Distance learning style
  - Course text
  - Self reading and practice required

7

Seminar Topics

- Introduction to java programming
- Selection
- Repetition
- Methods and Arrays
- Object Oriented Programming – Classes and objects
- Revision

Brief History of Java

- Original name was Oak. Later renamed to Java.
- First announced in the SunWorld Conference on May 1995.
- Its rise to fame came as the result of its niche in the World Wide Web technology.

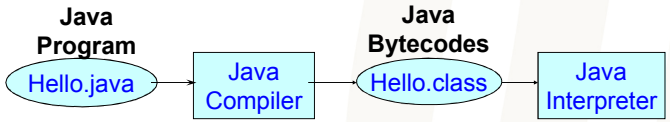
Software

- Java Development Kit
  - J2SE 7
  - http://www.oracle.com/technetwork/java/index.html
- Integrated Development Editor (IDE)
  - BlueJ 3.0.7
  - www.bluej.org
- Download and install

Java Compiler

- Each Java program is both **compiled** and **interpreted**.
- The **Java Compiler** translates a Java program into an intermediate language called **Java bytecodes** -- the platform-independent codes interpreter by the Java interpreter.

Java Programming cycle



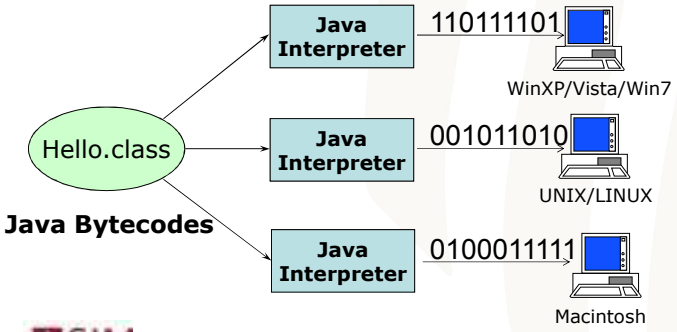
Step 1 : Write the Java source code

Step 2 : Compile the Java source code

Step 3 : Run the compiled Java program or class file

Java Interpreter

- The Java Interpreter parses the java bytecodes and sends it to the computer for execution.



A Simple Java Program

```
public class FirstJavaProgram
{
   public static void main(String[ ] args)
   {
        System.out.println("My first Java program!");
   }
}
```


Java Language Syntax

- Case Sensitive
  - Public not the same as public
- Braces { }
- Semi-colon ;
- Indent your program!

Output

- System.out.println( ) displays output
- E.g.

  System.out.println("My first java program");

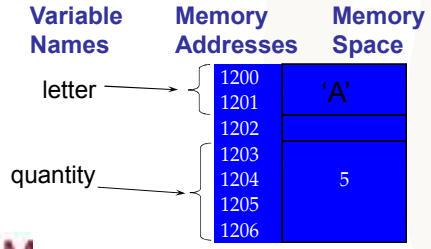
  Displays a line enclosed by " ". Cursor goes to the next line.

- What is the output of the following?

  System.out.println("My ");
  System.out.print("first ");
  System.out.println("java ");
  System.out.print("program ");

Variables

- A java program needs work space in memory to store data
- A **variable** is a name given to the memory cell(s) where the computer uses to store data.



Variables

- To use a variable, you must first declare it.
- A variable declaration consists of:
  - Data type
  - Name
- E.g: int quantity;
  double cost;
  String name;

Variable Name

- Case sensitive
- Cannot be any java keyword
- A name can be formed from uppercase and lowercase alphabets, digits, $, _
- No spaces, commas and symbols such as &, % and *
- Cannot start with a digit
- Are the following variable names acceptable?

count1	total-price	for	1stName
high_Score	maxDiscount		$amount

Java Keywords

abstract	boolean	break	byte
case	catch	char	class
const	continue	default	do
double	else	extends	false
final	finally	float	for
goto	if	implements	import
instanceof	int	interface	long
native	new	null	package
private	protected	public	return
short	static	super	switch
synchronised	this	throw	throws
transient	true	try	void
volatile	while		

Variables - Data Type

- Two different categories of data type
  - Primitive data types
    - int, double, float, char, boolean
  - Complex types (reference types)
    - Classes
    - String

Java's Primitive Types

There are 8 primitive types in Java:

- 4 whole number data types (byte, short, **int**, long)
- 2 real number data types (float, **double**)
- 1 character data type (**char**)
- 1 boolean data type (**boolean**)

String Type

- A reference type
- Strings are represented by a series of characters within double quotes.
- Examples:

```
String name = "John";
String address = "12 Java Street";
String phone = "91234567";
```


A Java Program with Variables

```
public class FirstJavaProgram {
   public static void main(String[ ] args) {
        String name = "John";
        System.out.println("My name is " + name);
   }
}
```

The + symbol appends the second string to the first

Input – Using Input Parameter

args[0]
"John"

```
public class Hello
{
        public static void main(String[ ] args)
        {
                String name = args[0];
                System.out.println("Hello " + name);
        }
}
```


Java's Primitive Types

Type	Storage	Range
byte	1 byte	-128 to 127
short	2 bytes	-32,768 to 32,767
int	**4 bytes**	**-2,147,483,648 to 2,147,483,647 (just over 2 billion)**
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	±1.40 E-45 to ±3.40 E+38, about 7 digits
double	**8 bytes**	**±4.94 E-324 to ±1.79 E+308, about 15 digits**

Java's Primitive Types

- Character
  - The character type is denoted by **char**.
  - **Character literals** are based on the Unicode encoding scheme and must appear between single quotes (Eg. 'A', 'B', …,'Z', 'a', …, 'z', '0', '1', …, '9'). It may also contain escape sequence (Eg. '\t', '\n', '\\', '\0').
- Boolean
  - The boolean type is denoted by **boolean**.
  - There are only 2 boolean literals : **true** and **false**.

Examples

- To declare a variable to store integer values
  int quantity;
- To declare a variable to store area of a circle
  double area;
- To declare a variable to store the grade of subject
  char grade;

Assigning values to Variables

- Variables hold values
- Use of assignment operator =
- E.g.

```
int  quantity = 5   ;
```

- Assigns the value from the right to the left
- Only 1 single variable is on the left of =
- Expression cannot appear on the left!
- Not the = sign in algebra!
  x = y+1 cannot be written as y+1 = x

Arithmetic (Binary) Operators

Operator	Use	Description
+	op1 + op2	Adds op1 and op2
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder after dividing op1 by op2

Expressions

- An expression is any combination of variables, constants and operators that can be evaluated to yield a result.
- The order of evaluation depends on the precedence of the arithmetic operators.
- Examples:
  price * quantity - discount
  (base + height) / 2
- Expressions are written on the right of assignment operator. E.g.
  double amount = price * quantity;
- Are the following statements valid?
  int x = 5;
  x + 2 = 2 * x - 5;

Example – Area of Circle Program

- To compute the area of a circle
- Given radius = 2.5

```
public class AreaOfCircle {
  public static void main(String[ ] args)   {
        double radius= 2.5;
        double area = 3.142 * radius * radius;
        System.out.println(area);
  }
}
```

String type vs Numeric types

- A numeric string “12” is not the same as number 12
- “12” + “34” does not result in “46” but “1234”
- + is used to append 2 strings instead of arithmetic add
- Therefore, numeric strings must be converted to their appropriate types before performing arithmetic operations on them

Converting Strings to Numeric

- To convert an integer string to an integer value

```
String sNum = “123”;
int num = Integer.parseInt(sNum);
```

- To convert a string with decimal to a double value

```
String sHt = “1.75”
double ht = Double.parseDouble(sHt);
```

Example - Using Input Parameter

- Use input parameter for radius

args[0]
“2.5”

```
public class AreaOfCircle {
  public static void main(String[ ] args)   {
        double radius= Double.parseDouble(args[0]);
        double area = 3.142 * radius * radius;
        System.out.println(area);
  }
}
```

35

Division Operator - /

- / operator computes the result when 2 numbers are divided
- E.g.

```
double x = 7.5 / 2;
```

(The result 3.25 will be stored in x)

- What about this?

```
double  y = 1 / 2;
```

Evaluating Expressions

- What is the result of x in the following 2 statements?

```
double x  = 1 / 2;
            int  int
x =          0
             int
```

x is assigned 0.0

```
double x  = 1 / 2.0;
            int  double
x =          0.5
             double
```

x is assigned 0.5

Modulus operator - %

- % computes the remainder when 2 numbers are divided
- E.g. int num = 14 % 3;
  (2 will be stored in num)

```
     4
3 ]14
   12
    2
```

Overloaded + operator

- If the operands are numeric, then the + operator is an arithmetic sum of the operands
- E.g. System.out.println(1.5 + 2.4);
  output is 3.9
- If any operand is a String type, then the + operator will convert the operands to String and the + operator is treated as concatenation
- E.g. System.out.println("1.5" + 2.4);
  output is ?

Incrementing a variable

- What is the output of the following?

```
int count = 10;
count  = count + 1;
System.out.println( count );
```

Special Assignment Operators

Operator	Use	Description
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

Arithmetic (Unary) Operators

Operator	Use	Description
++	op++	Increments op by 1; evaluates to value before incrementing
++	++op	Increments op by 1; evaluates to value after incrementing
--	op--	Decrements op by 1; evaluates to value before decrementing
--	--op	Decrements op by 1; evaluates to value after decrementing

Pre/Post Operator

- What is the result of x and y?

```
int x = 1;
int y = ++x;
```

- ++x is pre-increment. It increments x first (x becomes 2), then it's value is used to assign to y (y becomes 2)
- When used without the assignment =,

```
++x;
```

it is equivalent to x = x + 1;

Pre/Post Operator

- What is the result of x and y?

```
int x = 1;
int y = x++;
```

- x++ is post increment. That means the value of x is used to assign to y first (y is assign 1), then x is incremented (x becomes 2)
- When used without the assignment =,

```
x++;
```

it is equivalent to x = x + 1;

Exercise on Pre/Post Operators

- What is stored in variables x, y and z after the following statements?

```
int x = 10;
x++;
int y = x++;
int z  = --y;
```


Precedence of Operation

Postfix operators	[] . (params) expr++ expr--
unary operators	++expr —expr +expr -expr ~ !
creation or cast	new (type)expr
multiplicative	* / %
additive	+ -
relational	< > <= >= instanceof
equality	== !=
logical AND	&&
logical OR	||
conditional	? :
assignment	= += -= *= /= %= &= ^= < <= > >=

Casting

- Is this assignment valid?

  int x = 3.5;
- Trying to assign a double value 3.5 (8 bytes) to an integer variable (4 bytes) is not valid
- **Casting** allows one data type to be converted to another data type
- Two different types of casting : implicit casting and explicit casting

Implicit Casting

- The rule of thumb for assignments without a cast:

  double ← float ← long ← int ← short ← byte
- Example:

```
int x = 10;
double d = x;        // ok for int value -> double?
System.out.println( d );          // output?
```

Explicit Casting

- **Explicit Casting** is required when proceeding in the other direction.
- Example:

```
double d = 1.5;
int x = d;          // is this ok?

int x = (int)d;     // What is stored in x?
```


Using Math functions

- Java has many Math functions
- E.g. 1: 2 to the power of 5
  double result = Math.pow(2, 5);
- E.g. 2; Square root of 10
  System.out.println(Math.sqrt(10));
- The functions are in a Math class, therefore, the prefix Math is required beside the function names
- The functions return a value.
- Lookup Java class library for all available Math functions in the Math class

Formatting Output

- Use printf to format output
- E.g. To print x=2.6578 to 2 decimal places
  System.out.printf("value of x is %.2f", x);
  Output: value of x is 2.66
- Format of printf:
  System.out.printf(*fmtString, arguments*);
- Reference and more examples:

http://www.java2s.com/Tutorial/Java/0120__Development/0200__printf-Method.htm

Additional References

- A good reference for beginners
  - www.javaranch.com
  - www.javabeginner.com
- Java tutorials
  - http://download.oracle.com/javase/tutorial/
- Java API reference
  - http://download.oracle.com/javase/1,5.0/docs/api/