Elementary Java Programming

CSC02A2

Outline

Outline

1. The Basics
   - Identifiers, types and constants
   - Java Data Types
   - Operators and numerical type conversion
   - Character data
   - Ouput and Input
   - Hello World
2. Programming Constructs
   - Logical Operators
   - Conditional statements
   - Selection statements
   - Iterators
   - Flow control

The Basics

The Basics

Java programs consist of sets of objects which interact with each other via method invocation.

A Java class is created within a textual `.java` source file. The name of the outermost, public class must match the name of the Java source file.

Java source files are compiled into Java byte-code `.class` files by way of the *javac* program and run by way of the *java* program command. In order to be directly executed by the JVM a Java class must have a public main method.

Command-line arguments may be passed to the main method via a mechanism similar to that found in C++. Unlike C++ there is only one acceptable form of the main method.

Identifiers, types and constants

Identifiers in Java follow roughly the same rules as in C++.

- An identifier is a sequence of characters consisting of letters, digits, underscores and dollar signs.
- An identifier cannot start with a digit
- An identifier cannot be a reserved word.
- An identifier cannot be `true`, `false` or `null`.
- An identifier can be of any length.

In Java variables are either reference types or fixed length primitive types (see next slide).

Constants in Java are declared using the keywords ***final*** (which means that the variable's value cannot be altered) and ***static*** (which ensures that only one copy of the variable exists in memory). Constants are typically names in ALL_CAPS.

Java Data Types

Name	Range	Storage size	Wrapper class	Suffix
byte	$-2^7 \ldots 2^7 - 1$	8 bits	**Byte**	none
char	$0 \ldots 65535$	16 bits	**Character**	none
short	$-2^{15} \ldots 2^{15} - 1$	16 bits	**Short**	none
int	$-2^{31} \ldots 2^{31} - 1$	32 bits	**Integer**	none
long	$-2^{63} \ldots 2^{63} - 1$	64 bits	**Long**	L
float	$\pm 3.408235e{+}38$	32 bits	**Float**	F
double	$\pm 1.79769e{+}308$	64 bits	**Double**	D
Object	—	32 bits	—	—

Operators and numerical type conversion

- Operators behave in the same manner as C++.
- Operators in Java may not be overloaded.
- Numerical data types may be automatically converted by Java as follows
- If one operand is a **`double`** then the other is converted into a **`double`**
- Otherwise if one operand is a **`float`** the other is converted into a **`float`**
- Otherwise if one operator is a **`long`** the other is converted into a **`long`**
- Otherwise both operands are converted into type **`int`**

Take note

Data types may be manually converted via type casting. Beware of loss of information when moving from wider to shorter ranged data types and from higher to lower precision.

Character data

Java uses the Unicode character encoding scheme as opposed to the conventional **ASCII** scheme. The first block of Unicode values contains the entire **ASCII** table. The `\u` escape sequence may be used when setting a char variable's value:

```
char alpha = "\u03b1";
```

alpha now contains the character α

Escape characters need to be used in order to handle special characters:

- `\b` (backspace)
- `\t` (tab)
- `\n` (line feed)
- `\f` (form feed)
- `\r` (carriage return)
- `\\` (backslash)
- `\'` (single quote)
- `\"` (double quote)

`String` is an object type which may be used to store sequences of characters.

Basic and formatted output I

The **`System.out`** print stream contains:

- ***`print()`*** and ***`println()`*** for basic output to the console.
- ***`printf()`*** for formatted output.

```
1 // Basic output
2 System.out.println("String Expression");
3 // Formatted output
4 int d = 108;
5 boolean t = true;
6 System.out.printf("Favourite number is %d FM. The statement is %b", d, t);
```


Basic and formatted output II

Format specifiers are:

- %b - boolean
- %d - decimal
- %c – char
- %f – floating point
- %e – scientific notation
- %s – string.

In addition width and precision can be set as follows **%Width.Precisionf** e.g. **%10.2f** has a width of 10 and two decimal places.

Basic Input

Java provides a useful **`Scanner`** class for dealing with user input from devices such as the keyboard.

In order to convert between **`String`** and primitive data type values each primitive Wrapper class provides appropriate parsing methods.

Parsing the text may result in an **`Exception`**. **`Exception`** handling will be discussed in a future lecture.

Hello World class

```java
// This is an inline comment
// Importing Scanner class
import java.util.Scanner;
class HelloWorld
{
  /*
    This is a block comment over many
                      lines.
  */
  // Constant declaration
  public static final double PI = 3.14159265358979323846D;

  // Main method with arguments
  public static void main(String[] args)
  {
    // explicit type conversion
    int humble_pi = (int) PI;
    // Console output
    System.out.println("Hello World!");
    System.out.print("Whom shall I address?: ");
    // Wrap Scanner around Sysytem input
    Scanner scInput = new Scanner(System.in);
    // Read a line
    String strWho = scInput.nextLine();
    System.out.println("Why hello " + strWho);
  }
}
```


Compiling Hello World

To compile the class use the following command:

```
c:\Code>javac HelloWorld.java
```

Running the *javac* command will produce a **`HelloWorld.class`** class file. To run the program **`HelloWorld`** program use:

```
c:\Code>java HelloWorld
```


Programming Constructs

Logical Operators

Operator	Short circuit	Bit-wise	Description
NOT	!	!	Negation
AND	&&	&	Conjunction
OR	||	|	Disjunction
XOR	^	^	Exclusion

Conditional statements

```java
int value = 0;
if(booleanCondition)
{
  value = 1; // Perform truthful operation

}
else
{
  value = 2; // Perform false operation
}
// Tenary operator shorthand
value = booleanCondition ? 1 : 2;

// Dangling else
if(false)
  if(false)
   System.out.println("A");
else
  System.out.println("B");
// What is the output?
```


Selection statements

Switch statements allow different paths of execution for many different values without the clutter of many ***if else*** statements.

```
1  // Strings work in JDK7
2  switch(char|byte|short|int|enum|String)
3  {
4     case value0: //Code to handle value0;
5     break;
6     case value1: //Code to handle value1;
7     break;
8       ...
9    ...
10    case valueN: //Code to handle valueN;
11    break;
12    default: //Code to handle no matches;
13    break;
14 }
```


Iterators

```java
// While loop
while(booleanCondition)
{
   // loop body
}

// Do..while loop
do
{
  // loop body
} while(booleanCondition)

// For loop - Basic
for(initialisation; comparison; progression)
{
   // loop body
}

for(int k = 0; k < 10; ++k)
{
   System.out.println(k);
}

// For loop - Collections
for(String s : args)
{
   System.out.println(s);
}
```


Flow control

Iteration control is best performed by building the necessary logic into the loop's continuation condition. However sometimes necessary to end a loop or current loop iteration:

- ***break*** – immediately ends the current loop.
- ***continue*** – immediately ends the current loop iteration.

These keywords only operate in loops or switch statements.

For nested loops it is possible to use the labelled versions of the ***break*** and ***continue*** statements. This practice is discouraged as it dramatically reduces the readability of the code.

```
1 // How many  times will the outer loop run?
2 counting:
3 for(int k = 0; k < 10; ++k)
4   for(int j = 0; j < 10; ++j)
5     if(j == 9) break counting;
```