FUNDAMENTALS

Table of Contents

Java Fundamentals

This handout provides a brief overview of the Java syntax and conventions used in this course. It should be noted that not all of the keywords, available within Java, are described in this document.

Attributes

The table below compares a number of the attributes associated with the programming languages C++ and Java.

Attribute	C++	Java
Object-Oriented Principles	Allows the programmer to mix procedural and object-oriented programming approaches. This maintains backward compatibility with the C programming language and its libraries.	Only an object-oriented programming approach is allowed. As a result, global variables are not allowed.
Speed	It is a compiled language and the resultant executable is highly optimised for speed.	It is an interpreted language and so there is the overhead of the interpretation stage before the software is executed. This is generally not noticeable for routine applications but can become an issue for specific types of software such as operating systems and real-time systems.
Portability	The C++ programming language is designed to be used on any platform (operating system) but a compiler must be created for each platform. As a result , for each new platform, the program must be recompiled.	The Java programming language is interpreted and so is only compiled once to a specified format using byte codes. On each platform, a Java interpreter must be created but each interpreter can run the same compiled Java program. Therefore, a Java program is only compiled once regardless of the number of platforms (operating systems) being used.
Readability	Like C, C++ uses hard to read operators to represent specific object-oriented attributes.	Java uses distinct keywords that indicate their purpose. It

	These operators are used for a variety of purposes (overloading), depending on the context. For example, the ':' operator is used to signify one class inheriting from another. It is also used for other purposes in C++.	avoids the overloading of operators. For example, the keyword `extends` is used to show which class is being inherited from and is not used in any other context.

Keywords

In the table below, a comparison between the C++ and Java keywords is made (the list is not complete). Other Java keywords will be introduced as the course progresses.

C++	Java	Brief Description

`class Test`
`{`
`}`
	`class Test`
	`{`
	`}`
		Start of the class definition for `Test`.

`private:`
`void aMethod( void );`
`int a;`
	`private void aMethod();`
	`private int a;`
		Private method `aMethod` and private field `a`.

`protected:`
`void aMethod( void );`
`int a;`
	`protected void aMethod();`
	`protected int a;`
		Protected method `aMethod` and protected field `a`.

`public:`
`void aMethod( void );`
`int a;`
	`public void aMethod();`
	`public int a;`
		Public method `aMethod` and public field `a`.

`virtual aMethod( void ) = 0;`	`abstract void aMethod();`	Abstract method `aMethod`.
`#include <stdio.h>`	`import java.io.*;`	Including class libraries.
`class A : public B`	`class A extends B`	Class `A` inherits the attributes of class `B`. Single inheritance for Java and multiple inheritance for C++.
`class A : public B, public C`	`class A implements B, C`	Class `A` implements the interfaces `B` and `C`. In the C++ case, the classes `B` and `C` are abstract classes.
`namespace A { }`	`interface A { }`	Groups related methods (creates interfaces).
`namespace A { }`	`package A;`	Groups related classes together under a single name

		so as to prevent naming conflicts.
`const int a = 1;`	`final int a = 1;`	Constant `a` is assigned the value of `1`.

`aClass *a;`
 	
`a = new aClass;`
	`aClass a;`
	
	`a = new aClass();`
		The creation of an instance of the class `aClass`.
		
		In Java there are no pointers. Instead Java uses references for all class declarations.

`for( <initialize>; <test>; <action>  )`
`{`
`}`
	`for( <initialize>; <test>; <action>  )`
	`{`
	`}`
		The **`for`** loop.

`if( <expression> )`
`{`
`}`
	`if( <expression> )`
	`{`
	`}`
		The **`if`** statement.

`else`
`{`
`}`
	`else`
	`{`
	`}`
		The **`else`** statement.

`switch( <variable> )`
`{`
`    case <value>:`
`    {`
`        break;`
`    }`
 	
`    default:`
`    {`
`        break;`
`    }`
`}`
	`switch( <variable> )`
	`{`
	`    case <value>:`
	`        break;`
	
	`    default:`
	`        break;`
	`}`
		The **`switch`**, **`case`** and **`default`** statements.

`try`
`{`
`    <statements>`
`}`
`catch( anException e)`
`{`
`    <statements>`
`}`
	`try`
	`{`
	`    <statements>`
	`}`
	`catch( anException e)`
	`{`
	`    <statements>`
	`}`
		Creating exception handlers.

	`finally`
	`{`
	`    <statements>`
	`}`	

`while( <expression> )`
`{`
`}`
	`while( <expression> )`
	`{`
	`}`
		The **`while`** loop.

`do`
`{`
`} while( <expression> )`
	`do`
	`{`
	`} while( <expression> )`
		The **`do-while`** loop.

`continue`	`continue`	Skip the rest of the statements in the **`while`**, **`for`** or **`do-while`** loop and continue from the top of the loop.
`break`	`break`	Break out of **`while`**, **`for`** and **`do-while`** loops.

Operators

In the table below a comparison between the C++ and Java operators is made (the list is not complete).

C++	Java	Description
`()`	`()`	Parentheses do the following: • Group expressions (change associativity and precedence). • Isolate conditional expressions in **`if`**, **`while`**, **`for`**, **`switch`** and **`do-while`** statements. • Indicate method calls and parameters.
`[]`	`[]`	Indicate array subscripts.
`.`	`.`	Message passing and direct access to class fields.
`!`	`!`	Logical negation.
`++`	`++`	Increment.
`--`	`--`	Decrement.
`*`	`*`	Multiplication.
`/`	`/`	Division.
`%`	`%`	Modulus.
`+`	`+`	Addition.
`-`	`-`	Subtraction.

`>`	`>`	Greater than.
`>=`	`>=`	Greater than or equal to.
`<`	`<`	Less than.
`<=`	`<=`	Less than or equal to.
`==`	`==`	Equal to.
`!=`	`!=`	Not equal to.
`&&`	`&&`	Logical AND (conjunction).
`||`	`||`	Logical OR (disjunction).
`=`	`=`	Assignment.
`+=`	`+=`	Addition followed by assignment.
`-=`	`-=`	Subtraction followed by assignment.
`*=`	`*=`	Multiplication followed by assignment.
`/=`	`/=`	Division followed by assignment.
`%=`	`%=`	Modulus followed by assignment.

Operator Precedence

The Java operator precedence is given below.

```
highest     ()    []    .

            ++    --    ~     !

            *     /     %

            +     -

            >>    >>>   <<

            >     >=    <     <=

            ==    !=

            &

            ^

            |

            &&

            ||

lowest      =     +=    -=    *=    /=    %=
```

Primitive Types

Java guarantees the same size for each primitive type, no matter what operating system or hardware is being used. On the other hand, the sizes of the C++ types depend on the underlying hardware and the compiler used. However, the relative sizes are guaranteed in both cases. For example, the size of an `int` is always smaller than or equal to a `long`.

Note that since sizes vary in C++, the comparison below is from a functional view only.

C++	Java	Java Sizes
`char`	`char`	Java – 2 bytes (16 bits)
`No equivalent`	`byte`	Java – 1 byte (8 bits)
`short`	`short`	Java – 2 bytes (16 bits)
`int`	`int`	Java – 4 bytes (32 bits)
`long`	`long`	Java - 8 bytes (64 bits)
`float`	`float`	Java – 4 bytes (32 bits)
`double`	`double`	Java – 8 bytes (64 bits)
`bool`	`boolean`	Not available.

Case Sensitivity

Like its C++ counterpart, Java is case sensitive. For example, the words `astring` and `aString` are two separate identifiers. For those already experienced in C++, this should not pose any problems, however, some tips that can ease the situation are provided below.

- All Java keywords are in lowercase. For example, `while`, `for` and `class`.
- All the classes provided with the Java Software Development Kit (JDK) begin with an uppercase letter. For example, `String` and `System`.

Naming Conventions

The Java naming conventions used in this course are outlined below.

Attribute	Description	Example

Classes	Class names should be nouns, starting with the first letter of each word capitalized.	`class Test`
		`class GraphicsUtils`

Interfaces	Follows the same technique as classes.	`interface Visual`
		`interface ReadUtils`

Fields and variables	Start with the first letter as lowercase with each internal word starting with an uppercase letter.	`int x;`
		`char c;`
		`double steadyState;`

	For readability do not start variable names with underscore or $ characters.	

Methods	Method names should be verbs that indicate what action the method performs.
	
	Start with the first letter as lowercase with each internal word starting with an uppercase letter.
		`test();`
		`getWidth();`

Constants	Should be all uppercase with words separated by underscore _.	`final int MAX_WIDTH = 10;`
		`final int MAX_X_SQUARE = 2;`

Java Packages

In C++, programmers include the headers of the libraries that they wish to use and the C++ compiler then makes the necessary links. In Java, packages perform the same function. The difference is that Java does not allow the separation of the functions from the class definition like C++ (.h and .cpp files). Instead, all of the information for a class is in a .java file.

In Java, groups of related classes are placed in a Java package. The keyword used is `import`. Consider the following example.

```
import java.util.Scanner;
```

The Java `Scanner` class is being used and is located in the `java.util` package.

This line is included at the top of the code in the same way the #include statement is used in C++ to include header files.

Java Console Input and Output

Console input and output refers to the inputting and outputting of text at the command line. The built-in Java `System` class provides access to the standard error output (`err`), standard input (`in`) and standard output (`out`).

Standard Output

To output to the screen, use the Java `System.out` object (which is of type `PrintStream`) to access the standard output. The most commonly used functions are `print` and `println`. Both functions output text to the standard output but `println` adds a newline character at the end. Below is an example of its use.

```
class DemoOutput
{
	public static void main(String[] args) {
		System.out.print("Text without a new line.");
		System.out.println("Text with a new line.");
	}
}
```

Java Input

The `System.in` object is of type `InputStream` and is used to obtain input from the standard input. However, to access more advanced facilities, the `Scanner` class is used as follows.

```
import java.util.Scanner;

class DemoInput
{
      public static void main(String[] args) {
            // Scanner class accesses the standard input
            Scanner scan = new Scanner(System.in);
            String selection;

            // User enters the selection and presses Enter
            System.out.println("Enter your selection: ");
            selection = scan.nextLine();

            scan.close();
      }
}
```

The `Scanner` class provides functions that simplify the processing of the input.