C++ -> Java

Moving from C++ to Java

Executive Summary

- Introduces Java to a C++ programmer.
- Provides a roadmap for understanding advanced Java features.
- Links to Java resources.

Background

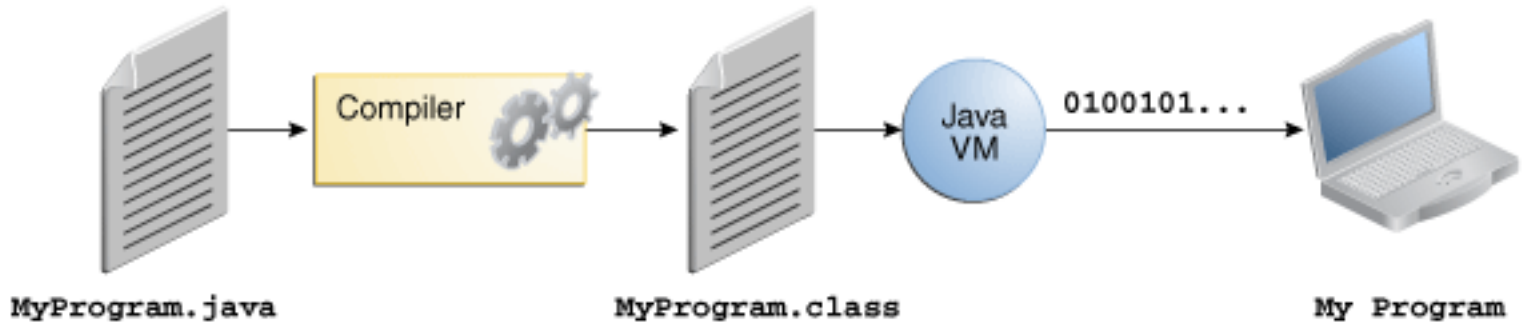
- Developed by James Gosling at Sun Microsystems.
- In 2011, it was the most popular programming language (as per TIOBE).
- Influenced by C and C++.
- Object Oriented programming language.
- Portable.
- Write once, run anywhere.

Java Virtual Machine

- Java source files are compiled into class files by the javac compiler.

- Unlike C++ object files, class files contains bytecode.

- The java launcher tool then runs the application with an instance of the JVM.



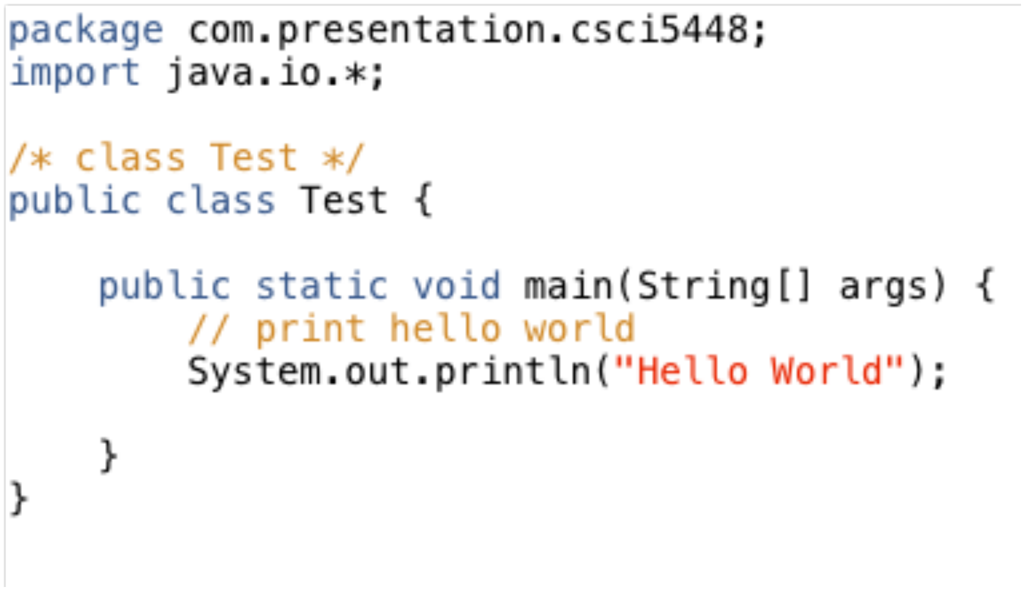
Downloading Java and IDE

- Where to download Java compiler: http://www.oracle.com/technetwork/java/javase/downloads/index.html

- Some popular IDEs:
  - Netbeans: http://netbeans.org/features/java/javase.html
  - Eclipse: http://www.eclipse.org/downloads/moreinfo/java.php
  - Jedit: http://www.jedit.org
  - Jcreator: http://www.jcreator.com

Hello World!

Source File: Test.java



```java
package com.presentation.csci5448;
import java.io.*;

/* class Test */
public class Test {

    public static void main(String[] args) {
        // print hello world
        System.out.println("Hello World");

    }
}
```

Comparing with “Hello World” in C++

- In C++, main is not a member function, however in Java it is. Java has no equivalent of non member functions.

- main method doesn't return a value in Java, so return type is void. C++ returns a value to indicate success or return error code to calling process.

Comparing with “Hello World” in C++

- In Java you can call System.exit(int return_code) to terminate and return a value.

- In C++, there is usually a header file and source file. However, in Java there is only one source file and name of source file always matches the name of the public class that it contains.

Java Import statement vs. C++ include

- No code is copied in Java from import statements.

- The import statements allow you to refer to the classes in the two packages without including the package name every time you use the class name.

- So, import statements can be compared to C++ *using*.

More on Import

- Just like *using* in C++, java import is not required, a class can be referred to by giving the path to the package:

```
java.util.Date today = new java.util.Date();
```

- Can import whole package or just the class.

```
 import java.util.*;    or
import java.util.Date;
```

Packages

- Java has packages. C++ doesn't.
- Packages allow you to group classes in a collection
- The main reason of packages is to guarantee uniqueness of class names.
- All standard java packages are inside java and javax package hierarchies.

Packages

- To include a class in a package, you need to put name of the package at the top of the source file.

- If name of the package is not included, it is assumed to be in the default package.

Packages

Naming convention:

- Package names are written in all lower case to avoid conflict with the names of classes or interfaces.
- Use reversed internet domain name to begin package names—for example, ***com.presentation.csci5448*** (in hello world example).

Comments

- Like C++, Java has block style (/* */) and single line (//) comments.

- In addition, comments in Java can be used for documentation. Javadoc utility program is used to extract the information and put it into an HTML file. (this is similar to doxygen tool - often used to generate C++ docs)

```
/**
 * This method returns the square of num.
 * @param num The value to be squared.
 * @return num squared.
 */
  public double square(double num) {
     return num * num;
  }
```

Only if…

In C++ you can do something like

```
int foo = 1;
if (foo) {
  // do something
}
```

- Won't work in Java because zero is not equivalent to false in Java and non-zero is not equivalent to true.

- In Java, conditional statements must evaluate to either true or false. Same applies *for, while* and *do while* statements in Java.

Arrays

In C++ arrays are primitive types. However, Array is a class in Java. You can use it's field - length (not function) on it to get it's size.

Syntax:

- type[] identifier; // preferred style by Java programmers.
- type identifier[]; // similar to C++

Examples
- int[] foo = new int[4];
- int foo[] = new int[4];
- int[] foo = {1, 2, 3, 4};

Arrays

- To copy an array, can't use an assignment operator, but can use Java.lang.System.arrayCopy(). It is similar to method memcpy provided by C++.

- In Java, operator [] is pre-defined to do bounds checking. If you try to access any index which is outside the defined range, Java will throw an “array index out of bounds” exception.

- Java provides *for each* loop to go through elements of the array. Example:

```
for (int element : a)
   System.out.println(element) // prints all the elements in
                               // the array a.
```

Strings

- In Java, the class similar to std::string is java.lang.String.
- String objects can be created with or without new:

```
String str = "abcd";
String str = new String("abcd");
```

- String classes are immutable. You cannot change the contents of the string after storage has been allocated.
- Java also provides a StringBuffer class which has methods that allow modification of strings.

Operators

- No operator overloading in Java.
- Cannot apply the arithmetic operators to object references.
- Some important C++ operators missing in Java: unary *, unary &, ->
- Does not support sizeof operator either.
- Does not support scope operator (::). The dot operator is used to call a method, regardless of whether it is a class or an instance variable.

Keywords

- No const in Java. Although, it is a reserved as a keyword in Java, it is not used and has no function.

- Final is the closest equivalent to const in Java. When applying to method or a class Java keyword final has a very different meaning that C++ const.

Keywords

Java does not let you use the access specifier as labels for a group of declarations.

Example:

C++:

```
 public:
void a1();
void a2();
```

Java:

```
public void a1();
public void a2();
```

Functions

In Java you cannot specify default value for arguments:

Example:

C++:

```
void foo(int i, int j=5)
{}
```

However, you can workaround this in Java:

```
void foo(int i, int j) {}
void foo(int i) { foo(i, 5); }
```

Types

- In Java, you cannot re-declare a variable in inner blocks:

```
public static void main(String[] args) {
int x = 1;
{   // inner loop
 int x = 4; // This will give an error in Java.
}
}
```

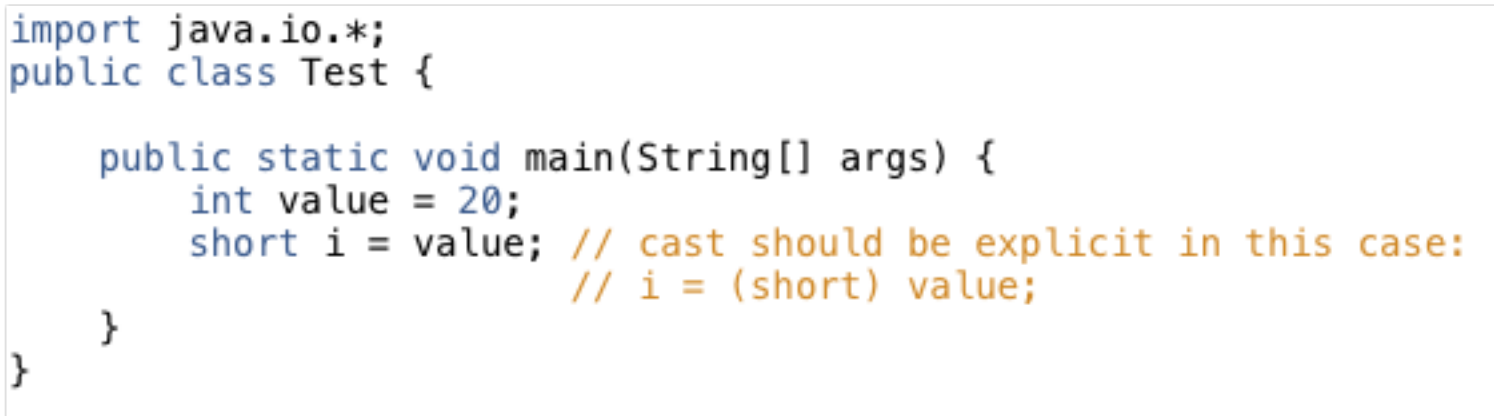
- C++ supports both signed and unsigned integral types. However, integral types are only signed in Java.

Types

- Java provides wrapper classes for all primitive types. Example, Integer for int.
- Java platform defines the size of the primitive types and the JVM gives them the same representation on all the machines.
- No *typedef* in Java.
- Equivalent of *bool* in Java : *boolean*

Casts

- Java doesn't implicitly cast a smaller type into a larger type (C++ usually gives a warning).



```
import java.io.*;
public class Test {

    public static void main(String[] args) {
        int value = 20;
        short i = value; // cast should be explicit in this case:
                         // i = (short) value;
    }
}
```

```
Test.java:9: error: possible loss of precision
 short i = value;
           ^
  required: short  found:    int
 1 error
```

Classes and Objects

Tomato Tamato

- C++ - member functions
  Java - method

- C++ - data member
  Java - field

- C++ - base class
  Java - superclass

- C++ - derived class
  Java - subclass

Classes and Objects

- In C++, you don't have to use classes and can code procedurally, but Java you have to use classes and hence encourages object oriented programming

- Java does not support struct, enum or union.

- No initializer lists in Java. So you can't do something like:

```
public class A {
        private a1_, a2_;
        A::A(int a1, int a2) : a1_(a1), a2_(a2) {…}
};
```

Classes and Objects

- All Java objects are constructed on the heap and a constructor must be combined with new.

  In C++, you can create an object by:

  ```
  Myobject obj;
  ```

  However in Java, obj will refer to an object **only** when it is initialized with new or copy constructed.

  ```
  Myobject obj = new Object;  OR
  obj = anotherObj;
  ```

- Class definitions don't end with a semicolon in Java.

Classes and Objects

- Objects are passed by reference ALWAYS.
- Think of Java references as similar to C++ pointers. But, without the book keeping.
- Automatic garbage collection, so NO destructors.

Abstract Class

In C++, a class is said to be abstract when you initialize one or more function to 0 (pure virtual function). In Java, *abstract* keyword is used to specify an abstract class.

```
abstract class Animal {
    public getName() {
        return name;
    }
    public abstract String getGroup();
    private name;
}
```

Abstract Class

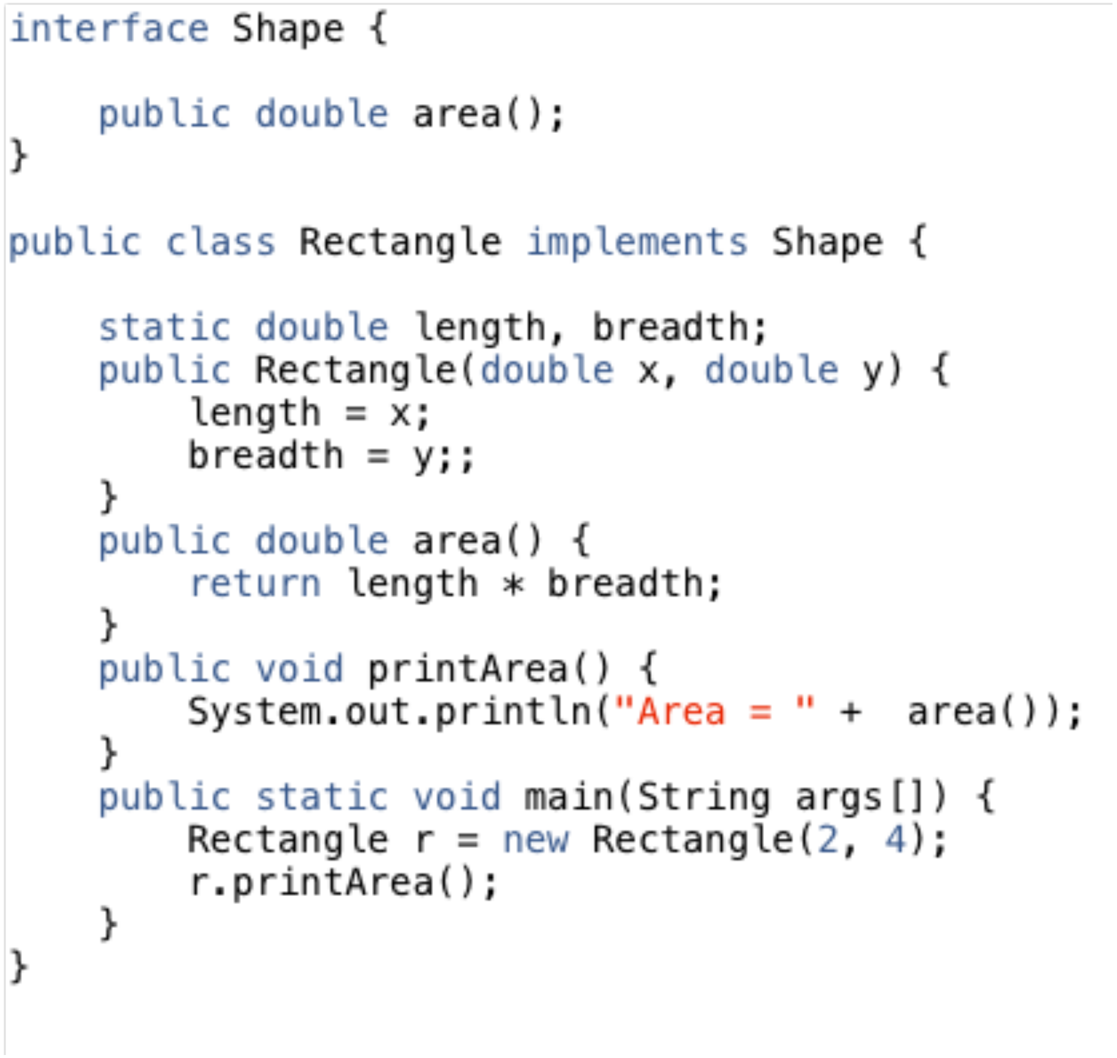
- Just like C++, you can’t make an instance of an abstract class.

- To make a method abstract, use *abstract* keyword in function definition.

- Subclass can either implement abstract methods or leave them undefined. In later case, the class has to be defined abstract.

Interfaces

- C++ doesn't have anything similar.
- "This is what all classes that *implement* this particular interface will look like." (Reference: Thinking in Java)
- Not a class, but set of requirements (contract). Keyword interface is used to define an interface. Keyword implements is used by class that implements it.
- Completely abstract, no implementation at all. It cannot be instantiated.

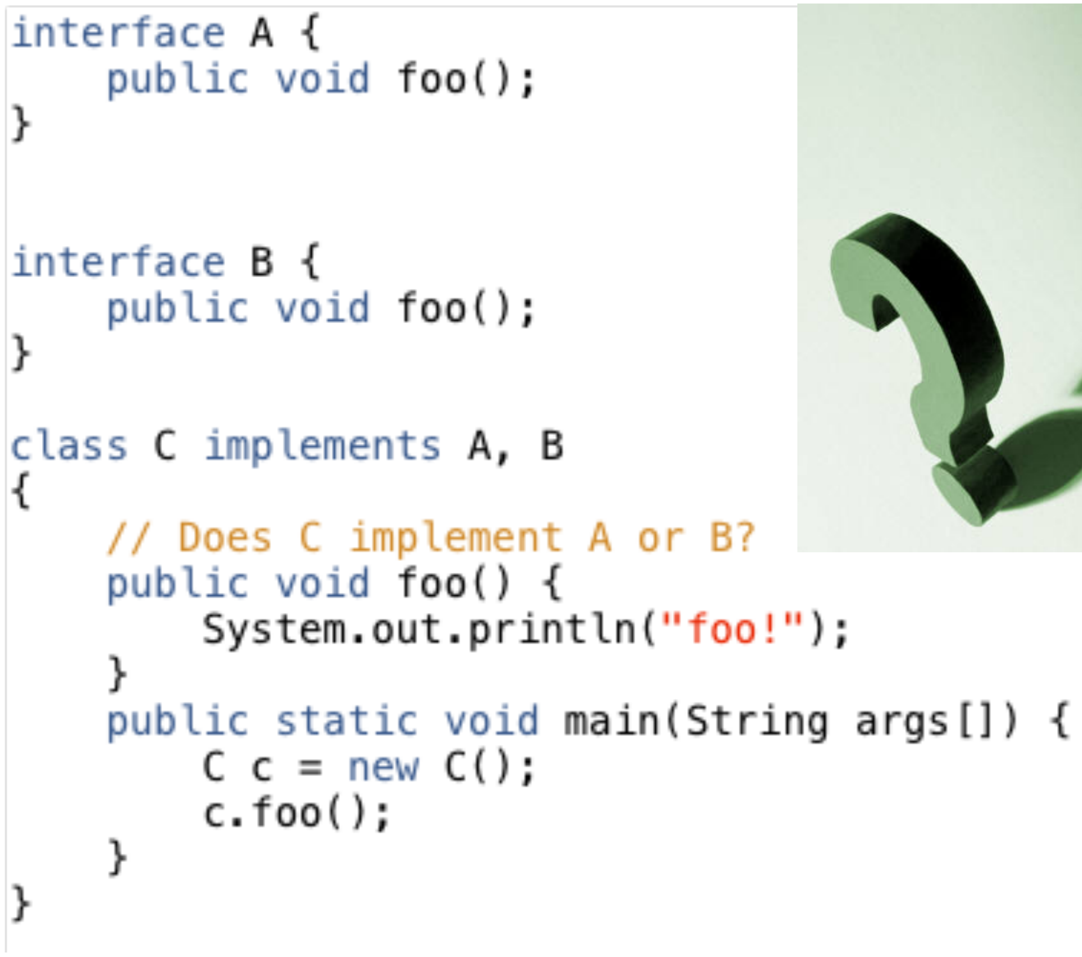


```java
interface Shape {

    public double area();
}

public class Rectangle implements Shape {

    static double length, breadth;
    public Rectangle(double x, double y) {
        length = x;
        breadth = y;;
    }
    public double area() {
        return length * breadth;
    }
    public void printArea() {
        System.out.println("Area = " +  area());
    }
    public static void main(String args[]) {
        Rectangle r = new Rectangle(2, 4);
        r.printArea();
    }
}
```

Interfaces

- A class can implement more than one interfaces.
- An interface can extend one or more interfaces (using extends).
- A class that implements an interfaces has to provide implementation of **ALL** the methods defined in an interface.

Interfaces



```java
interface A {
    public void foo();
}


interface B {
    public void foo();
}

class C implements A, B
{
    // Does C implement A or B?
    public void foo() {
        System.out.println("foo!");
    }
    public static void main(String args[]) {
        C c = new C();
        c.foo();
    }
}
```

Inheritance

- Cannot override **and** overload methods in inherited classes.

- Java doesn't support multiple inheritance.

- No protected or private inheritance. All inheritance in Java is public.

- Inheritance syntax is different from that of C++.

  C++ uses :

  Java uses *extends*.

Inheritance

- Classes can be **final** in Java. That implies that they cannot be inherited. The wrapper classes for primitive types are final in Java.

- In C++ you can call a method from any class in inheritance hierarchy. However, in Java you can only call immediate super class using keyword super.

Access Specifiers

- public access - provides unlimited access to all classes.

- protected access - gives unlimited access to all classes in same package. For classes outside the package, they can inherit the fields but cannot access them directly. A class in a different package and doesn't inherit, has no access to protected specifiers.

Access Specifiers

- default access - if no access specifier is mentioned, the access level is package. At package level, you have unlimited access to all the classes in the same package. But, classes outside the package cannot access the classes.

Note: C++ doesn't have the rule that access specifier of the overridden method is same or less restrictive.

Exceptions

- An exception object is always an instance of a class derived from *Throwable*. In C++, exception could be of any type.

- Types of Exceptions:
  - RuntimeException

    - Similar to C++ logic_error class.

    - Programming error.

    - Unchecked exception: Any exception that derives from RuntimeException.

Exceptions

- Non RuntimeException:

  - Similar to C++ runtime_error class.

  - I/O error, EOF, etc.

  - Checked Exceptions (a method **must** declare all checked exceptions).

Exceptions

- throws

```
void myException() throws ArrayIndexOutOfBoundsException {
        …
}
```

- Compilers gives an error if the throws clause doesn't list uncaught exception. In C++ this is optional

- Overriding method cannot list more exception types in the throws clause than the overridden method.

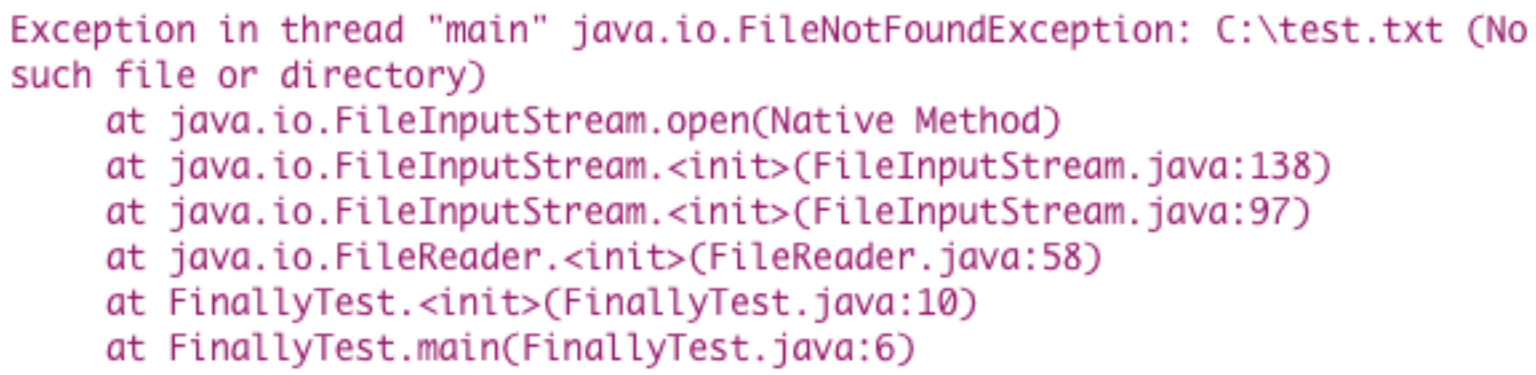
Exceptions

- Java has a *finally* clause, C++ doesn't.
    - Cleanup resource allocations like database connection.
    - *finally* executes whether or not the exception is caught.

```java
import java.io.*;

public class FinallyTest {

    public static void main(String args[]) throws IOException {
        FinallyTest f = new FinallyTest("C:\\test.txt");
    }

    public FinallyTest(String fileName) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(fileName));
        try {
            String line = null;
            while ( (line = reader.readLine()) != null ) {
            }
        }
        finally {
            reader.close();
        }
    }
}
```

Exceptions

Output :



```
Exception in thread "main" java.io.FileNotFoundException: C:\test.txt (No
such file or directory)
	at java.io.FileInputStream.open(Native Method)
	at java.io.FileInputStream.<init>(FileInputStream.java:138)
	at java.io.FileInputStream.<init>(FileInputStream.java:97)
	at java.io.FileReader.<init>(FileReader.java:58)
	at FinallyTest.<init>(FinallyTest.java:10)
	at FinallyTest.main(FinallyTest.java:6)
```

Java interpreter outputs the stack trace when an exception is un-caught.

Reflection

- To inspect classes, interfaces, fields and methods at runtime, without knowing the names of the classes, methods etc. at compile time.
- The class that holds this information is called ***Class*** - not a typo! Example: Class c = myObj.getClass();
- Similar to type_info in C++, but type_info doesn't give as much information.
- A good overview of Reflection:
  http://www.ibm.com/developerworks/library/j-dyn0603/

ANT (Another Neat Tool)

- Preferred build tool used for compiling Java programs.
- James Duncan Davidson, Ant's original author, states that Ant is "like make without make's wrinkles"
- By default ant uses **build.xml** as the name for a build file.
- To compile, package and run:
  - ant compile
  - ant jar
  - ant run

```
<project>

<target name="clean">
<delete dir="build"/>
</target>

<target name="compile">
<mkdir dir="build/classes"/>
<javac srcdir="src" destdir="build/classes"/>
</target>

<target name="jar">
<mkdir dir="build/jar"/>
<jar destfile="build/jar/Test.jar" basedir="build/classes">
<manifest>
<attribute name="Main-Class" value="com.presentation.csci5448.Test"/
>
</manifest>
</jar>
</target>

<target name="run">
<java jar="build/jar/Test.jar" fork="true"/>
</target>

</project>
```

JUNIT

- JUnit is a simple framework to write repeatable tests.
- JUnit assumes that all test methods can be executed in an arbitrary order. Therefore tests should not depend on other tests.

Installation:

http://www.junit.org/

Getting Started with Junit:

http://junit.sourceforge.net/doc/cookbook/cookbook.htm

References

- Core Java Volume 1 by Cay S. Horstmann and Gary Cornell
- Thinking in Java by Bruce Eckel
- Heads first Java by Kathy Sierra and Bert Bates
- http://ant.apache.org
- http://www.junit.org/
- http://docs.oracle.com/javase/tutorial/