

Java Classes

Introduction to the Java Programming Language

Produced
by

Eamonn de Leastar
edeleastar@wit.ie

Department of Computing, Maths & Physics
Waterford Institute of Technology

<http://www.wit.ie>

<http://elearning.wit.ie>



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LAIRGE



Essential Java

⊕ Overview

- ⊕ Introduction
- ⊕ Syntax
- ⊕ Basics
- ⊕ Arrays

⊕ Classes

- ⊕ Classes Structure
- ⊕ Static Members
- ⊕ Commonly used Classes

⊕ Control Statements

- ⊕ Control Statement Types
- ⊕ If, else, switch
- ⊕ For, while, do-while

⊕ Inheritance

- ⊕ Class hierarchies
- ⊕ Method lookup in Java
- ⊕ Use of this and super
- ⊕ Constructors and inheritance
- ⊕ Abstract classes and methods

Interfaces

⊕ Collections

- ⊕ ArrayList
- ⊕ HashMap
- ⊕ Iterator
- ⊕ Vector
- ⊕ Enumeration
- ⊕ Hashtable

⊕ Exceptions

- ⊕ Exception types
- ⊕ Exception Hierarchy
- ⊕ Catching exceptions
- ⊕ Throwing exceptions
- ⊕ Defining exceptions
- Common exceptions and errors

⊕ Streams

- ⊕ Stream types
- ⊕ Character streams
- ⊕ Byte streams
- ⊕ Filter streams
- ⊕ Object Serialization

Overview

- ⊕ **Classes in Java**
 - ⊕ What are classes?
 - ⊕ Defining classes
 - ⊕ .java files
 - ⊕ Packages and access level
 - ⊕ .jar files and classpath
 - ⊕ Fields, methods, and constructors
- ⊕ **Static fields and methods**
 - ⊕ Defining and using static fields
 - ⊕ Defining and using static methods
- ⊕ **Commonly used classes in Java**
 - ⊕ Object class
 - ⊕ String and String Buffer classes
 - ⊕ Class and System classes

Road Map

- ⊕ **Classes in Java**
 - ⊕ What are classes?
 - ⊕ Defining classes
 - ⊕ .java files
 - ⊕ Packages and access level
 - ⊕ .jar files and classpath
 - ⊕ Fields, methods, and constructors
- ⊕ **Static fields and methods**
 - ⊕ Defining and using static fields
 - ⊕ Defining and using static methods
- ⊕ **Commonly used classes in Java**
 - ⊕ Object class
 - ⊕ String and String Buffer classes
 - ⊕ Class and System classes

How to Define Java Class?

- ⊕ Java class is defined with using class keyword
 - ⊕ Class name follows the keyword, and by convention starts with capital letter
 - ⊕ For example Policy, Client, House, etc.
- ⊕ Class access level must be specified before the class keyword

```
public class Policy
{
    ...
}
```

Class Modifiers

- ⊕ Class Modifiers identifies visibility of the class
- ⊕ There are two access modifiers for a class:
 - ⊕ public
 - ⊕ Identifies that any other class can reference defined class
 - ⊕ Not specified
 - ⊕ Identifies that only classes defined in the same package can reference defined class
 - ⊕ It is default access level modifier

.java Files

- ⊕ Java classes are contained in .java files
 - ⊕ One file can contain one public class
 - ⊕ One file can contain more than one non-public classes
- ⊕ The file name is the same as the class name contained in the file

```
package org.tssg.demo.models;  
  
public class Policy  
{  
    ...  
}
```

Policy.java

Package

- ⊕ Package groups related classes
 - ⊕ Classes are usually related by their functionality, for example domain classes, testing classes, etc.
- ⊕ Package is identified using package keyword
- ⊕ Package is unique identifier for a class
 - ⊕ Two classes with a same name cannot be in the same package
 - ⊕ Different packages can contain same class names

```
package org.tssg.pim;
```


Referencing Classes

- ⊕ A class must be fully referenced every time when used outside of its package
- ⊕ Full qualifier for a class is used
 - ⊕ package name + class name

```
package org.tssg.demo.tests;  
  
public class PolicyTester  
{  
    org.tssg.demo.models.Policy policy;  
    ...  
    policy = new org.tssg.demo.models.Policy();  
}
```

Import Statement

- ⊕ Used to identify which classes can be referenced without fully identifying them
 - ⊕ Specified with import keyword
 - ⊕ Can specify a class, or all classes from a package

```
package org.tssg.demo.tests;
import org.tssg.models.Policy;

public class PolicyTester
{
    Policy policy;
    ...
    policy = new Policy();
}
```

Compiling Classes

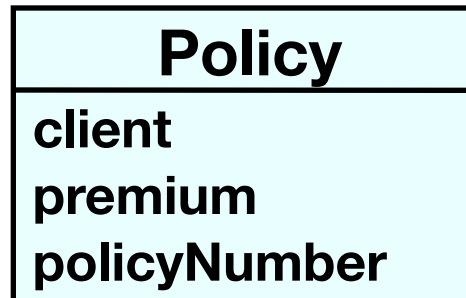
- ⊕ When writing Java class, the source code is stored in .java files
- ⊕ When compiling Java classes, compiled code is stored in .class files
- ⊕ Compiled Java classes from the same package are compiled into the same directory
 - ⊕ The directory name matched package name

Classpath and .jar files

- ⊕ Classpath allows Java Virtual Machine to find the code
 - ⊕ CLASSPATH environment variable is used to indicate the root of where packages are
 - ⊕ Packages are subdirectories under the root
- ⊕ Compiled Java classes can be packaged and distributed in Java Archive (.jar) files
 - ⊕ Packages in the .jar file are replaced with directories

What are Fields?

- ⊕ Object state is implemented through fields
- ⊕ Fields are defined at the class level
 - ⊕ All instances of the same class have the same fields
 - ⊕ Fields values can be different from instance to instance



Defining Fields

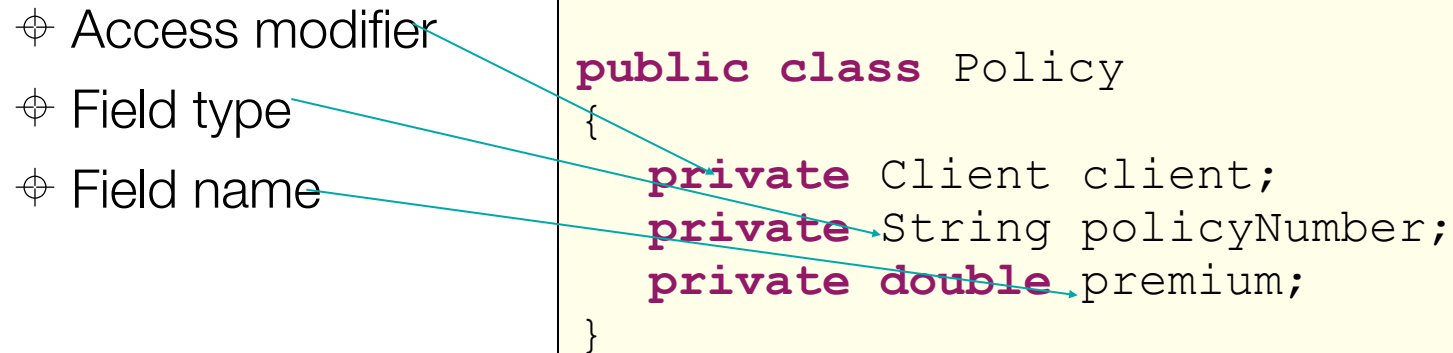
⊕ A field definition consists of:

⊕ Access modifier

⊕ Field type

⊕ Field name

```
package org.tssg.demo.models;  
  
public class Policy  
{  
    private Client client;  
    private String policyNumber;  
    private double premium;  
}
```



Initializing Fields

- ⊕ Fields are initialized when new instance of a class is created
- ⊕ Primitive type fields get a default value
 - ⊕ Numeric primitives are initialized to 0 or 0.0
 - ⊕ boolean primitives are initialized to false
 - ⊕ Reference type fields are initialized to null as they do not yet reference any object
- ⊕ Fields can also be explicitly initialized when declared

Initializing Fields Explicitly

- ⊕ Possible when declaring fields
- ⊕ Constructors are generally used for initializing fields

```
package org.tssg.demo.models;

public class Policy {
    private Client client = new Client();
    private String policyNumber = "PN123";
    private double premium = 1200.00;
}
```


Field Access Modifier

⊕ There are four different modifiers:

⊕ public

⊕ Allows direct access to fields from outside the package where class is defined

⊕ protected

⊕ Allows direct access to fields from within the package where class is defined

⊕ default

⊕ Allows direct access to fields from within the package where class is defined and all subclasses of the class

⊕ private

⊕ Allows direct access to fields from class only

Methods

- ⊕ Methods represent behavior of an object
 - ⊕ All instances of the same class have same methods defined and understand same messages
- ⊕ When a message is sent to an object, method that corresponds to that message is executed
 - ⊕ Methods represent implementation of messages

getters()/setters()

- ⊕ To allow access to private fields, getter and setter methods are commonly used
 - ⊕ Getters return fields values
 - ⊕ Setters set fields values to passed parameters

Policy
client premium policyNumber
getClient getPremium getPolicyNumber setClient setPremium setPolicyNumber

Defining Methods

- ⊕ Methods are defined with:
 - ⊕ Access modifier, same as for fields
 - ⊕ Return type
 - ⊕ Method name
 - ⊕ Parameters, identified with type and name

```
package org.tssg.demo.models;

public class Policy
{
    ...
    public void setClient(Client aClient)
    {
        ...
    }
}
```

Constructors

- ⊕ Special methods used for creating instances of a class:
 - ⊕ access modifier
 - ⊕ same name as the class
 - ⊕ manipulate new instance

```
package org.tssg.demo.models;

public class Policy
{
    ...
    public Policy()
    {
        setClient(new Client());
        setPolicyNumber("PN123");
        setPremium(1200.00);
    }
}
```

Using Constructors

- ⊕ Use `new` before class name to create an instance of a class

```
package org.tssg.demo.models;

public class Policy
{
    ...
    public Policy(Client aClient, String policyNumber, double premium)
    {
        setClient(aClient);
        setPolicyNumber(policyNumber);
        setPremium(premium);
    }
}
```

```
Policy policy = new Policy(new Client(), "PN123", 1200.00);
```

Policy Class Implementation

```
package org.tssg.demo.models;

public class Policy
{
    private Client client;
    private String policyNumber;
    private double premium;

    public Policy(Client aClient, String policyNumber, double premium)
    {
        setClient(aClient);
        setPolicyNumber(policyNumber);
        setPremium(premium);
    }

    public Client getClient()
    {
        return client;
    }

    public void setClient(Client aClient)
    {
        this.client = aClient;
    }
    //... other getters and setters..
}
```

Road Map

- ⊕ Classes in Java
 - ⊕ What are classes?
 - ⊕ Defining classes
 - ⊕ .java files
 - ⊕ Packages and access level
 - ⊕ .jar files and classpath
 - ⊕ Fields, methods, and constructors
- ⊕ Static fields and methods
 - ⊕ Defining and using static fields
 - ⊕ Defining and using static methods
- ⊕ Commonly used classes in Java
 - ⊕ Object class
 - ⊕ String and String Buffer classes
 - ⊕ Class and System classes

What are Static Fields?

- ⊕ Static fields represent data shared across all instances of a class
 - ⊕ There is only one copy of the field for the class
 - ⊕ Modification to the static field affects all instances of the class
- ⊕ Static fields are also known as class variables
- ⊕ Some of the static fields usages include:
 - ⊕ Constants
 - ⊕ Implementation of singleton pattern

Declaring Static Fields

- ⊕ Declared by using the static keyword
- ⊕ Java constants are declared as static final fields
 - ⊕ Modifier final indicates that field value cannot be changed

```
public class Count
{
    public static String INFO = "Sample Count Class";
    public final static int ONE = 1;
    public final static int TWO = 2;
    public final static int THREE = 3;
}
```

Accessing Static Fields

⊕ Static field can be accessed:

⊕ Directly

```
System.out.println(Count.ONE);
```



Console

1

⊕ Indirectly

```
Count count = new Count();  
System.out.println(count.INFO);
```



Console

Sample Count Class

Static Methods

- ⊕ Define behavior related to the class, not individual instances
 - ⊕ Defined by using the static keyword
 - ⊕ Commonly used for accessing static fields
 - ⊕ Getter and setter methods

```
public class Count
{
    private static String INFO = "Sample Count Class";
    public final static int ONE = 1;
    public final static int TWO = 2;
    public final static int THREE = 3;
    public static String getInfo()
    {
        return INFO;
    }
}
```

Using Static Methods

- ⊕ Static methods can be also accessed by instance or class

```
System.out.println(Count.getInfo());
```



Console
Sample Count Class

```
Count count = new Count();  
System.out.println(count.getInfo());
```



Console
Sample Count Class

Road Map

- ⊕ Classes in Java
 - ⊕ What are classes?
 - ⊕ Defining classes
 - ⊕ .java files
 - ⊕ Packages and access level
 - ⊕ .jar files and classpath
 - ⊕ Fields, methods, and constructors
- ⊕ Static fields and methods
 - ⊕ Defining and using static fields
 - ⊕ Defining and using static methods
- ⊕ Commonly used classes in Java
 - ⊕ Object class
 - ⊕ String and String Buffer classes
 - ⊕ Class and System classes

Package java.lang

- ⊕ It is a core package in Java
- ⊕ When classes from this package are referenced there is no need for import statement
- ⊕ Contains core set of classes such as:
 - ⊕ Object
 - ⊕ String
 - ⊕ StringBuffer
 - ⊕ System
 - ⊕ Class

Object Class

- ⊕ Object class is the top of the class hierarchy in Java
 - ⊕ Every class inherits from Object class
 - ⊕ Defines some default behavior that is mainly overridden in subclasses
- ⊕ Commonly overridden methods from Object class are:
 - ⊕ toString()
 - ⊕ equals()
 - ⊕ hashCode()
 - ⊕ clone()

Method equals()

- ⊕ Meant to return whether or not two objects are equal
 - ⊕ Default implementation in Object class returns whether or not are objects identical
 - ⊕ The == operator is used
 - ⊕ Overriding method allows to change the equality criteria
 - ⊕ For example two policies are the same if they have the same client, same policyNumber and same premium

Example equals() method

- ⊕ An example of overriding the equals() method in the Policy class
 - ⊕ Two policies are equal if their policy numbers are equal

```
public boolean equals(Object anObject)
{
    if (other == this)
    {
        return true;
    }
    if (other == null)
    {
        return false;
    }
    if (getClass() != other.getClass())
    {
        return false;
    }
    Policy policy = (Policy)anObject;
    return getPolicyNumber().equals(policy.getPolicyNumber());
}
```

Method hashCode()

- ⊕ Used by collections, primarily HashMap and HashSet
 - ⊕ Returns an int for indexing
 - ⊕ Hash codes must be identical for objects that are equal
 - ⊕ For the Policy class implementation of the hash code method could be:

```
public int hashCode ()  
{  
    return getPolicyNumber().hashCode();  
}
```

String Class

- ⊕ Used for manipulating constant set of characters
- ⊕ Literals are String instances that cannot be changed, and have fixed size

```
String greeting = "Hello" + ", do you like my hat?";  
                //"Hello, do you like my hat?"  
String hello = greeting.substring(0,5);                //"Hello"  
String upercase = hello.toUpperCase();                //"HELLO THERE!"  
boolean isEqual = hello.equals("HELLO");            //false  
boolean isEqual1 = hello.equalsIgnoreCase("HELLO"); //true
```

StringBuffer Class

- ⊕ Used for strings that can change
- ⊕ Allows for adding, replacing and deleting characters
 - ⊕ When characters are added size increases
 - ⊕ StringBuffer object knows about its length and capacity
 - ⊕ length indicates how many characters it has
 - ⊕ capacity indicates how many characters it can currently hold

Using StringBuffer Class

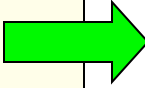
- ⊕ Typical buffer manipulation includes appending, replacing, inserting and deleting characters

```
StringBuffer buffer = new StringBuffer();  
buffer.append("Hello");  
buffer.append(", do you");  
buffer.insert(13, " like my hat?");  
System.out.println(buffer);  
buffer.replace(0, 5, "Hi");  
System.out.println(buffer);  
buffer.delete(2, buffer.length() - 1);  
buffer.replace(buffer.length() - 1,  
                buffer.length(),  
                "!");  
System.out.println(buffer);
```

Console



```
Hello, do you like my hat?
```



```
Hi, do you like my hat?
```



```
Hi!
```

System Class

- ⊕ Provides an access to system functions through its static protocols
 - ⊕ It is not possible to create instances of System class
 - ⊕ Defines static methods and fields

```
System.out.println("Hello, do you like my hat?");
```

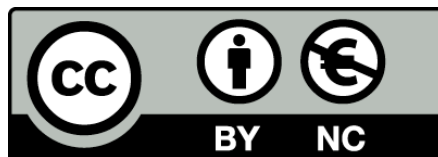


```
Hello, do you like my hat?
```

Console

Summary

- ⊕ Classes in Java
 - ⊕ What are classes?
 - ⊕ Defining classes
 - ⊕ .java files
 - ⊕ Packages and access level
 - ⊕ .jar files and classpath
 - ⊕ Fields, methods, and constructors
- ⊕ Static fields and methods
 - ⊕ Defining and using static fields
 - ⊕ Defining and using static methods
 - ⊕ Singleton pattern
- ⊕ Commonly used classes in Java
 - ⊕ Object class
 - ⊕ String and String Buffer classes
 - ⊕ Class and System classes



Except where otherwise noted, this content is licensed under a Creative Commons Attribution-NonCommercial 3.0 License.

For more information, please see <http://creativecommons.org/licenses/by-nc/3.0/>

