

Java Classes

An introduction to the Java Programming Language

Produced Eamonn de Leastar (edeleestar@wit.ie)
by: Dr. Siobhan Drohan (sdrohan@wit.ie)



Waterford Institute of Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE



Centre for
Technology-Enhanced Learning

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: 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?

Class access level / modifier

If not specified, only
classes in the
package can access it.

Class keyword


Class name

(note convention:
start with a capital
letter and non-plural)

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

.java Files

- ⊕ can contain one public class
- ⊕ can contain more than one non-public classes (e.g. Android framework adopts this approach).
- ⊕ Filename is the same as the public class.



Policy.java

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

Package

Package identifier



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

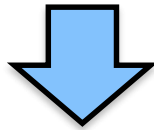
A package groups related classes e.g. testing, utilities, etc.

A package is a unique identifier for a class; two classes in a package cannot be called the same name

Referencing Classes / Import Statement

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

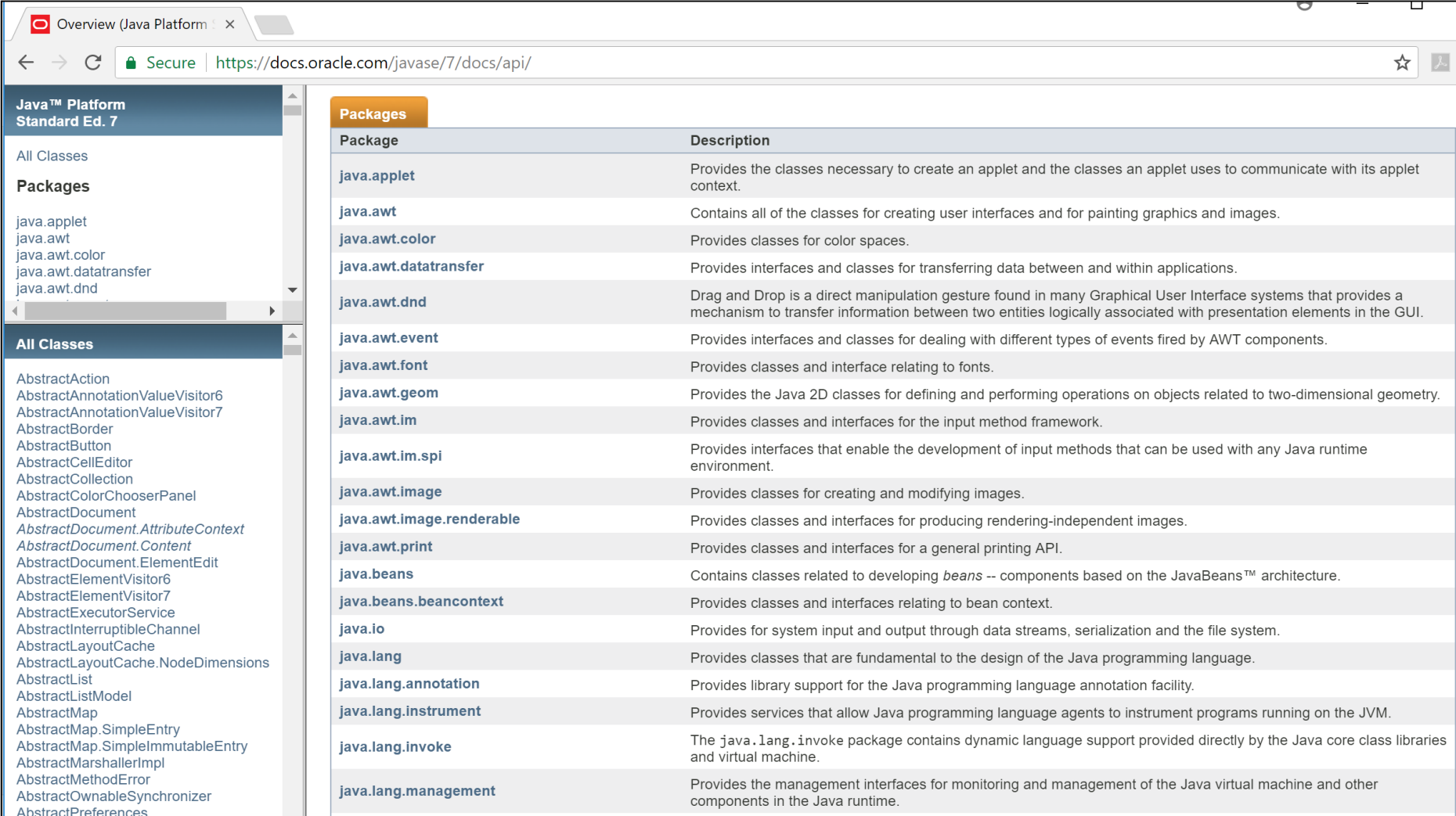
Approach 1



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

Approach 2
(preferred)

Some Java API packages...



The screenshot shows the Oracle Java Platform Standard Ed. 7 API documentation page. The browser address bar displays <https://docs.oracle.com/javase/7/docs/api/>. The page is titled "Overview (Java Platform Standard Ed. 7)".

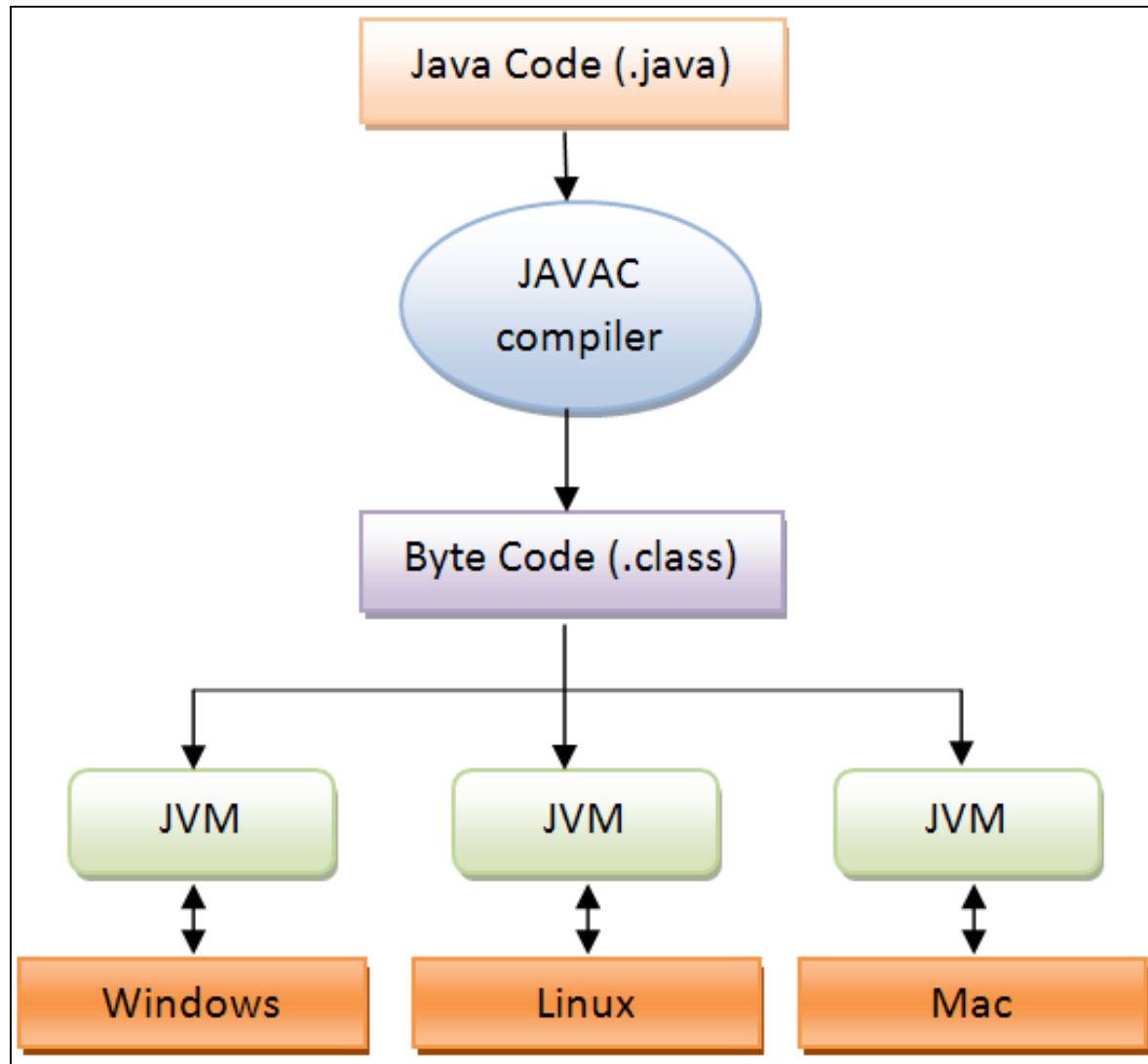
The left sidebar contains a navigation menu with the following sections:

- Java™ Platform Standard Ed. 7
- All Classes
- Packages
 - java.applet
 - java.awt
 - java.awt.color
 - java.awt.datatransfer
 - java.awt.dnd
- All Classes
 - AbstractAction
 - AbstractAnnotationValueVisitor6
 - AbstractAnnotationValueVisitor7
 - AbstractBorder
 - AbstractButton
 - AbstractCellEditor
 - AbstractCollection
 - AbstractColorChooserPanel
 - AbstractDocument
 - AbstractDocument.AttributeContext
 - AbstractDocument.Content
 - AbstractDocument.ElementEdit
 - AbstractElementVisitor6
 - AbstractElementVisitor7
 - AbstractExecutorService
 - AbstractInterruptibleChannel
 - AbstractLayoutCache
 - AbstractLayoutCache.NodeDimensions
 - AbstractList
 - AbstractListModel
 - AbstractMap
 - AbstractMap.SimpleEntry
 - AbstractMap.SimpleImmutableEntry
 - AbstractMarshallerImpl
 - AbstractMethodError
 - AbstractOwnableSynchronizer
 - AbstractPreferences

The main content area displays a table of Java API packages and their descriptions:

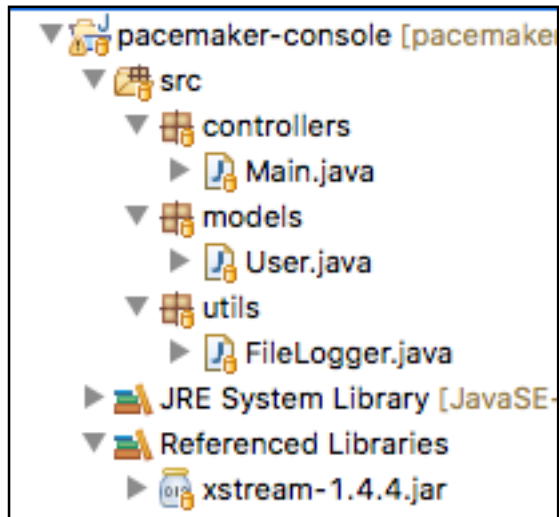
Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing <i>beans</i> -- components based on the JavaBeans™ architecture.
java.beans.beancontext	Provides classes and interfaces relating to bean context.
java.io	Provides for system input and output through data streams, serialization and the file system.
java.lang	Provides classes that are fundamental to the design of the Java programming language.
java.lang.annotation	Provides library support for the Java programming language annotation facility.
java.lang.instrument	Provides services that allow Java programming language agents to instrument programs running on the JVM.
java.lang.invoke	The <code>java.lang.invoke</code> package contains dynamic language support provided directly by the Java core class libraries and virtual machine.
java.lang.management	Provides the management interfaces for monitoring and management of the Java virtual machine and other components in the Java runtime.

Compiling Classes

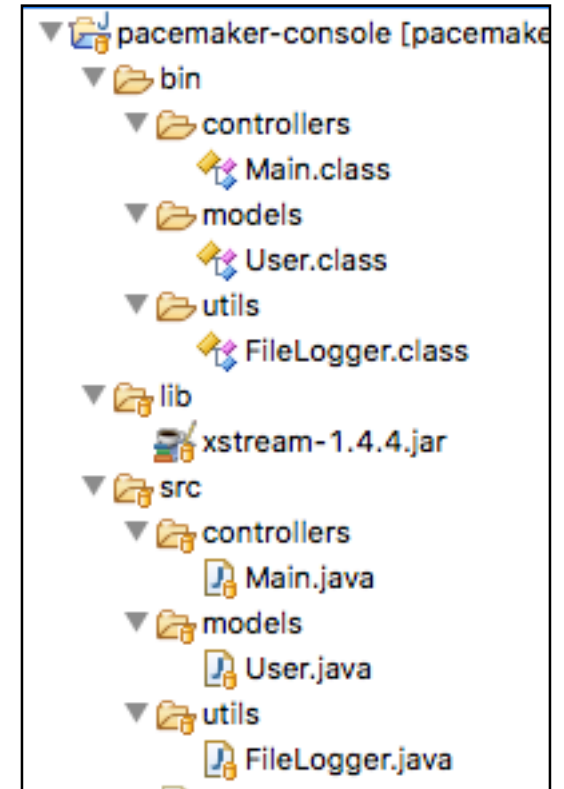


Compiling Classes

Eclipse Package Explorer View



Eclipse Navigator View



Logical



Physical



Java classes - in src directory - are compiled into the same folder in a bin directory

Classpath and .jar files



CLASSPATH is an environment variable that specifies the location of the classes and packages for the JVM.



Compiled Java classes can be packaged and distributed in Java Archive (.jar) files.
(packages become directories in the file).

BuildPath in Eclipse

Project → Properties

The image displays two overlapping screenshots of the Eclipse IDE's 'Properties for GymApp' dialog, specifically the 'Java Build Path' tab.

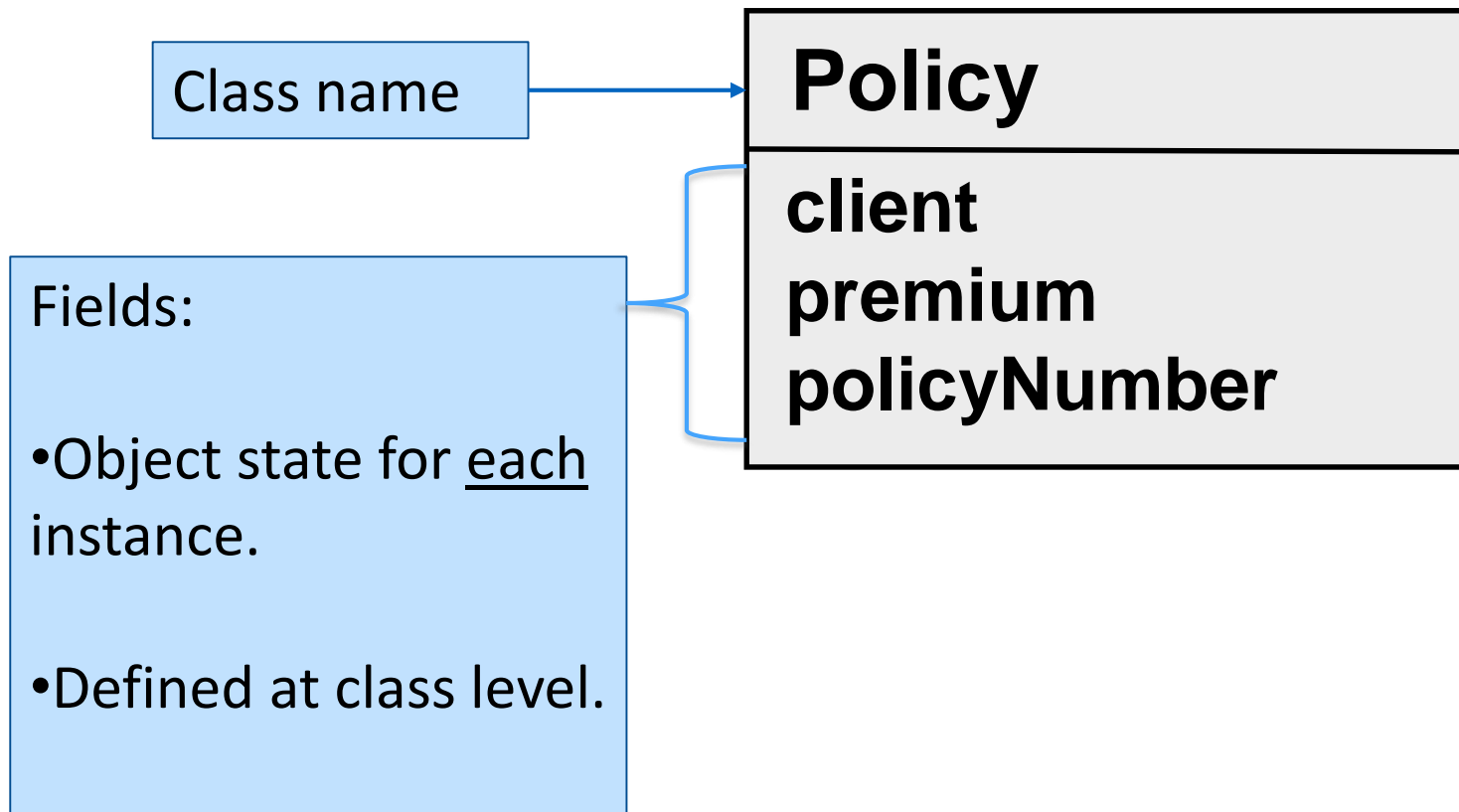
Top Screenshot:

- Left Panel:** A list of project properties including Resource, Builders, Coverage, Git, **Java Build Path** (selected), Java Code Style, Java Compiler, Java Editor, Javadoc Location, Project References, Run/Debug Settings, Structure101, Task Tags, and Validation.
- Right Panel:** The 'Java Build Path' tab with sub-tabs: Source, Projects, Libraries, and Order and Export. The 'Source' sub-tab is active, showing 'Source folders on build path:' with a list containing 'GymApp/src'. Buttons on the right include 'Add Folder...', 'Link Source...', and 'Remove'.

Bottom Screenshot:

- Left Panel:** The same list of project properties, with 'Java Build Path' selected.
- Right Panel:** The 'Libraries' sub-tab is active, showing 'JARs and class folders on the build path:' with a list containing 'xstream-1.4.8.jar - GymApp/lib' and 'JRE System Library [JavaSE-1.8]'. Buttons on the right include 'Add JARs...', 'Add External JARs...', 'Add Variable...', 'Add Library...', 'Add Class Folder...', 'Add External Class Folder...', 'Edit...', 'Remove', and 'Migrate JAR File...'. At the bottom right are 'Apply', 'OK', and 'Cancel' buttons.

What are Fields?



Defining Fields

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

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

Access modifier

Field type

Field name

Initializing Fields Explicitly

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

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

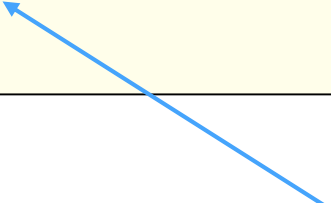
Primitive type fields get a default value:

- numerics = 0
- boolean = false

Initializing Fields Explicitly

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

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



Unless explicitly initialized, reference type fields are initialized to null.

Constructors are often used to initialize objects

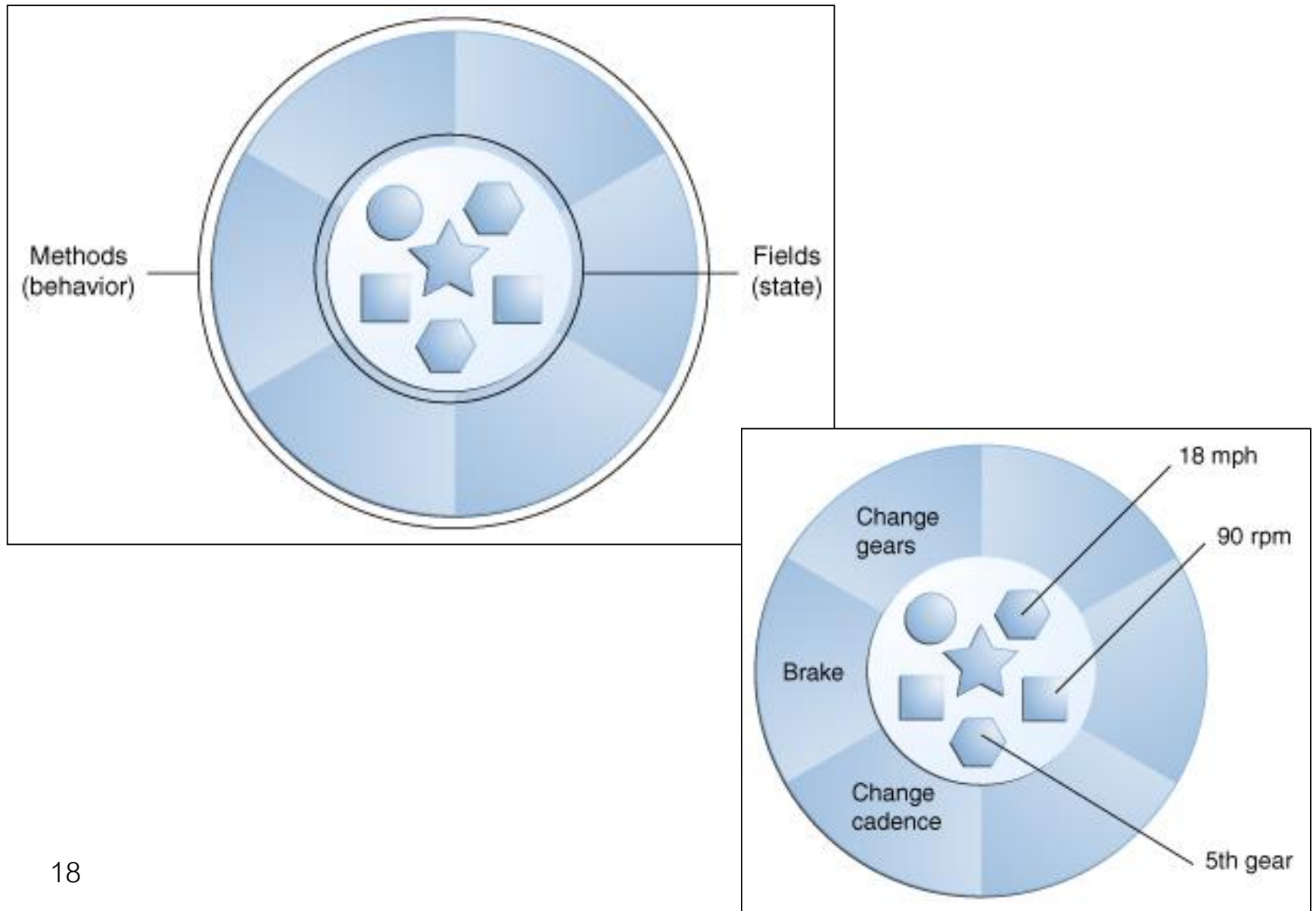
Primitive type fields get a default value:

- numerics = 0
- boolean = false

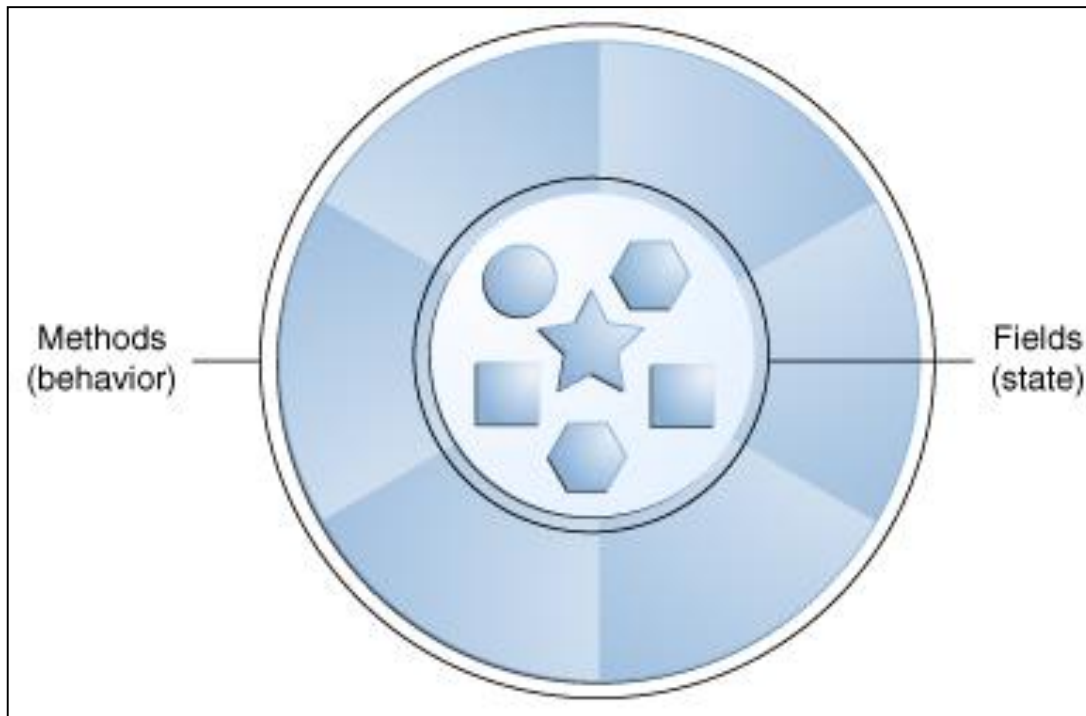
Field Access Modifier

Access Modifiers	Default	private	protected	public
Accessible inside the class	yes	yes	yes	yes
Accessible within the subclass inside the same package	yes	no	yes	yes
Accessible outside the package	no	no	no	yes
Accessible within the subclass outside the package	no	no	yes	yes

Methods



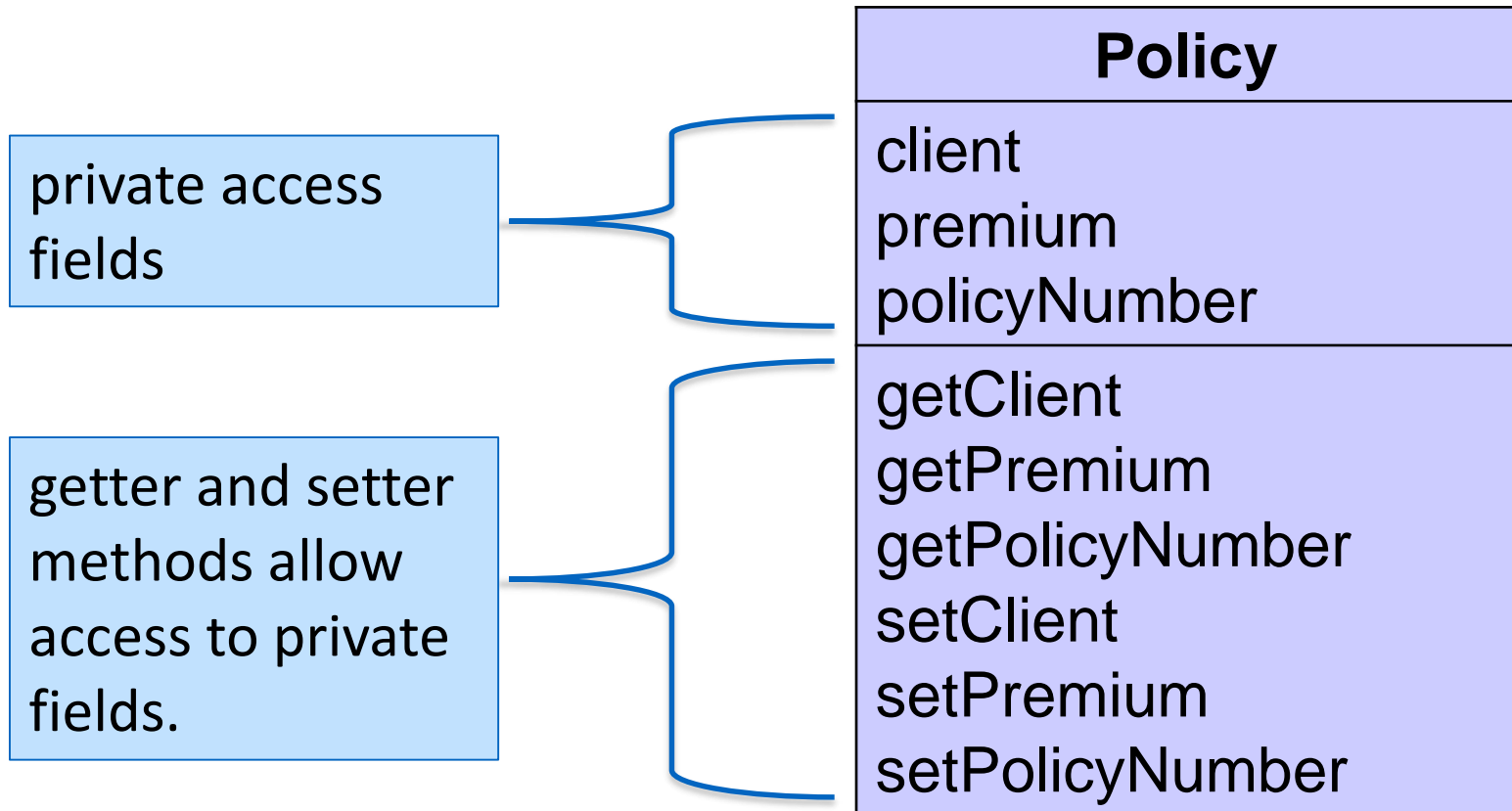
Methods



- All instances of the same class have same methods defined

- When a message is sent to an object, the method that corresponds to that message is executed i.e.
 - ⊕ Methods represent implementation of messages

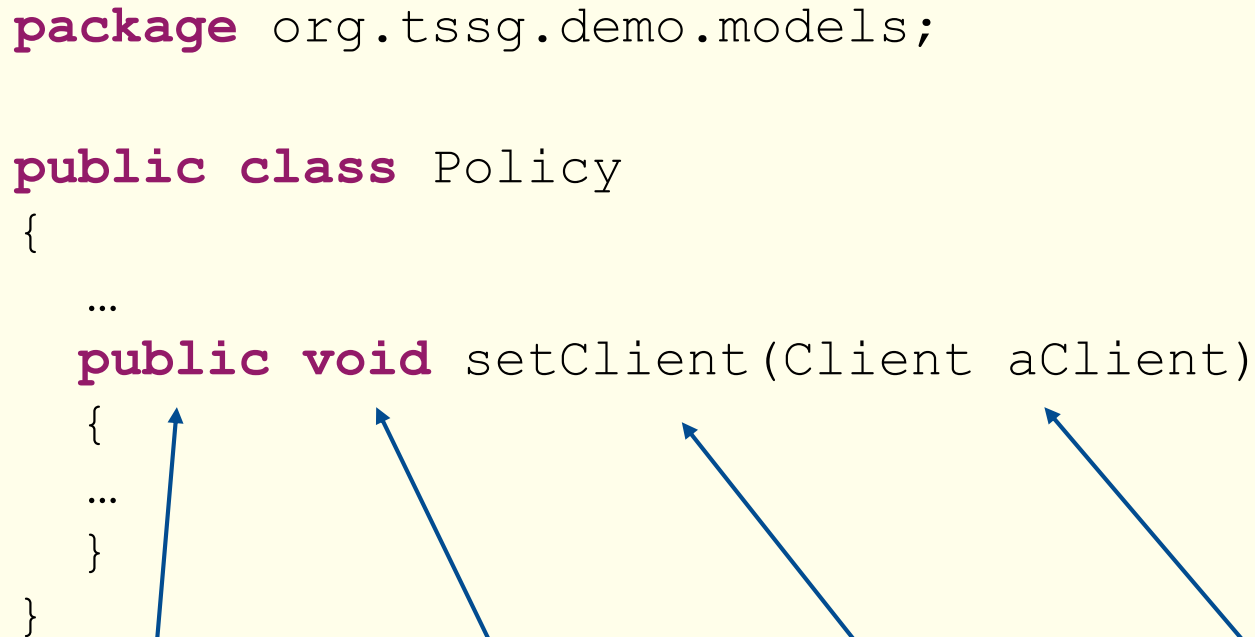
getters()/setters()



Defining Methods

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

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



Access modifier

Return type

Method name

Parameters

Constructors

Access
modifier

No return
type

Same name as
the class

Special method used
for creating
instances of a class.

Also initializes the
instance to a starting
state

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

```
public class Policy  
{
```

```
...
```

```
public Policy()  
{
```

```
    setClient(new Client());  
    setPolicyNumber("PN123");  
    setPremium(1200.00);
```

```
}
```

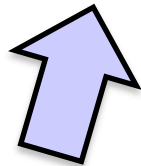
```
}
```

Constructors

```
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: Sample 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;
    }

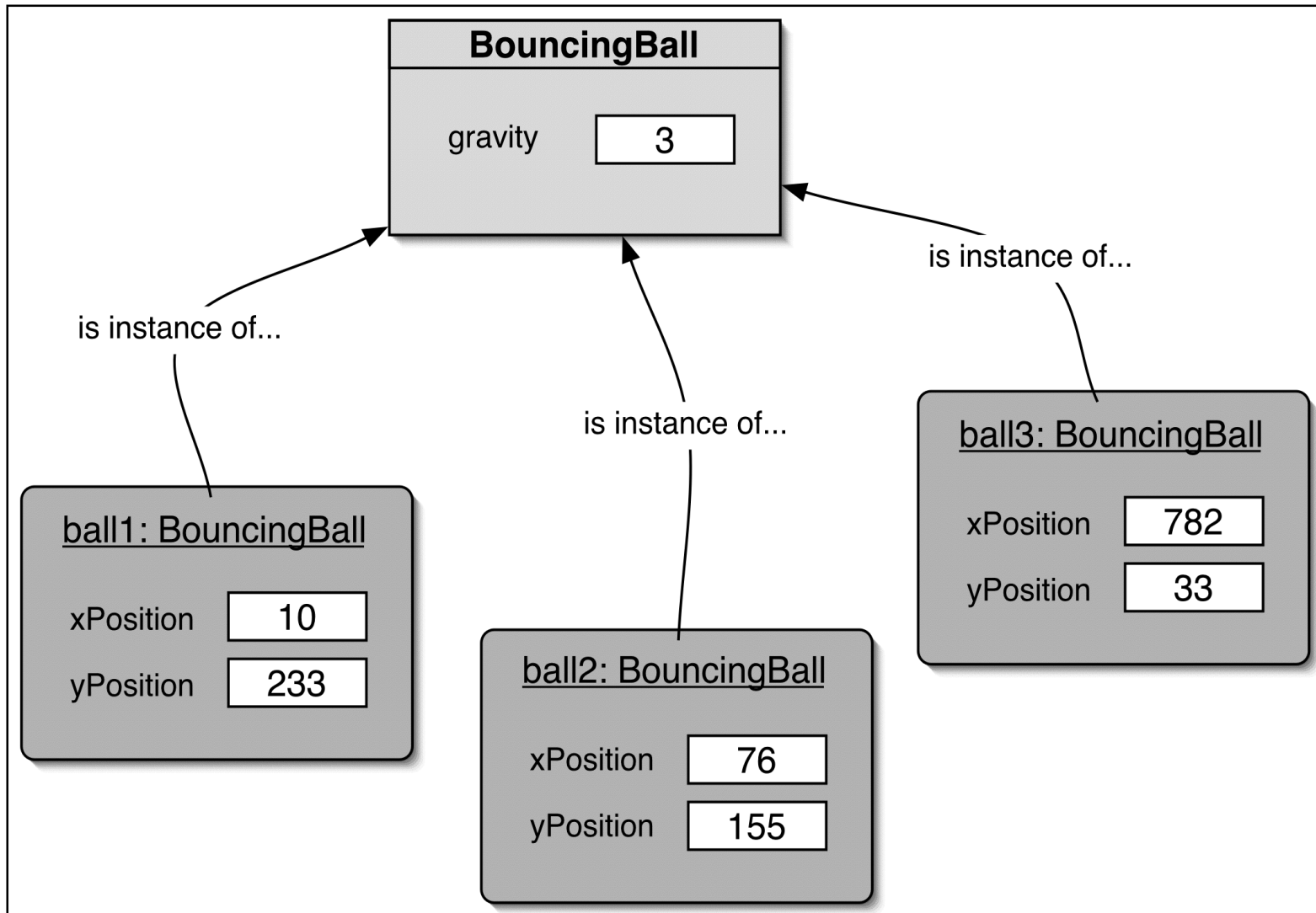
    public void setClient(Client aClient)
    {
        this.client = aClient;
    }
    //... other getters and setters..
}
```


Overview: 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?

```
private static int gravity;
```



Declaring Static Fields

Static field

Constant
fields i.e.
final

```
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

```
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;
}
```

Direct access

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



Console

1

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



Console

Sample Count Class

Indirect access

Static Methods

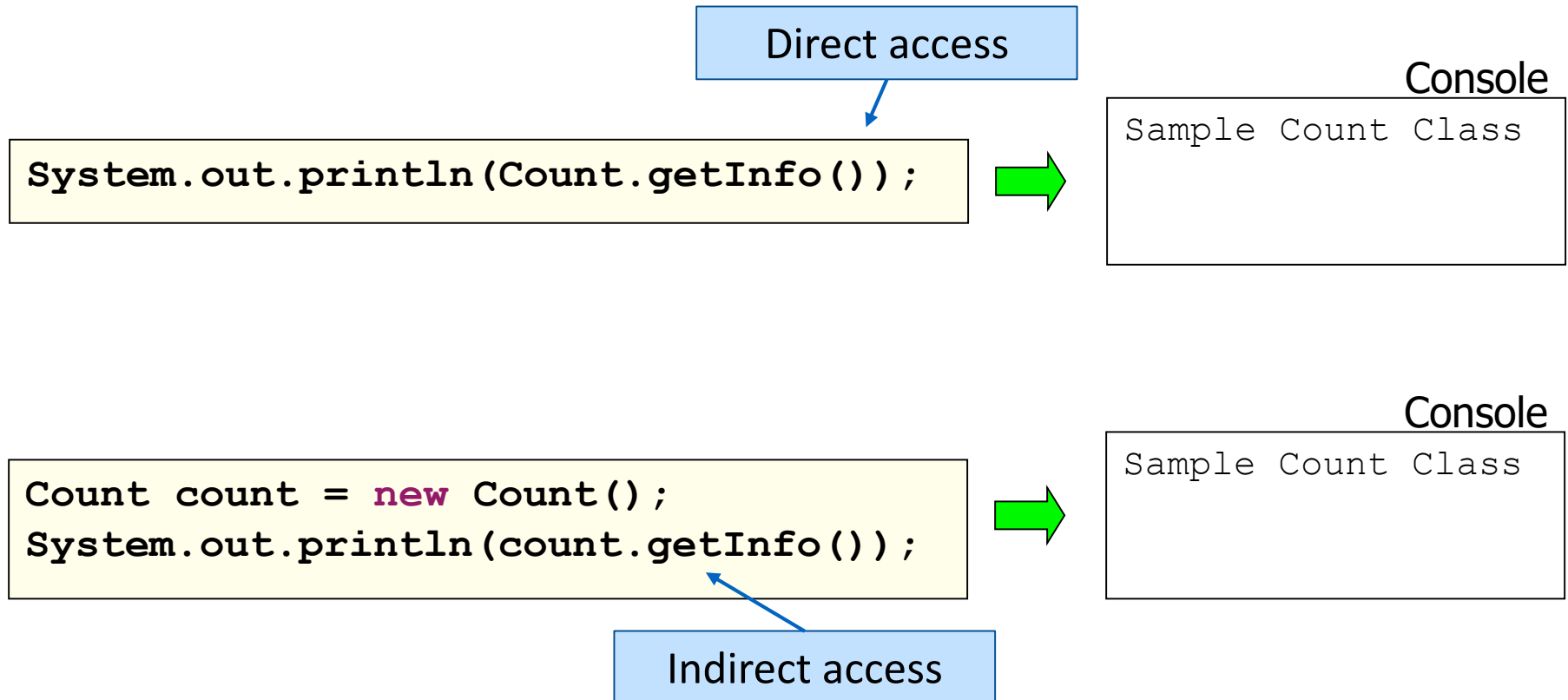
Static method
for behavior
related to the
class, not
instances.

```
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;
    }
}
```

Commonly used for
accessing static fields.

Using Static Methods




Overview: 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

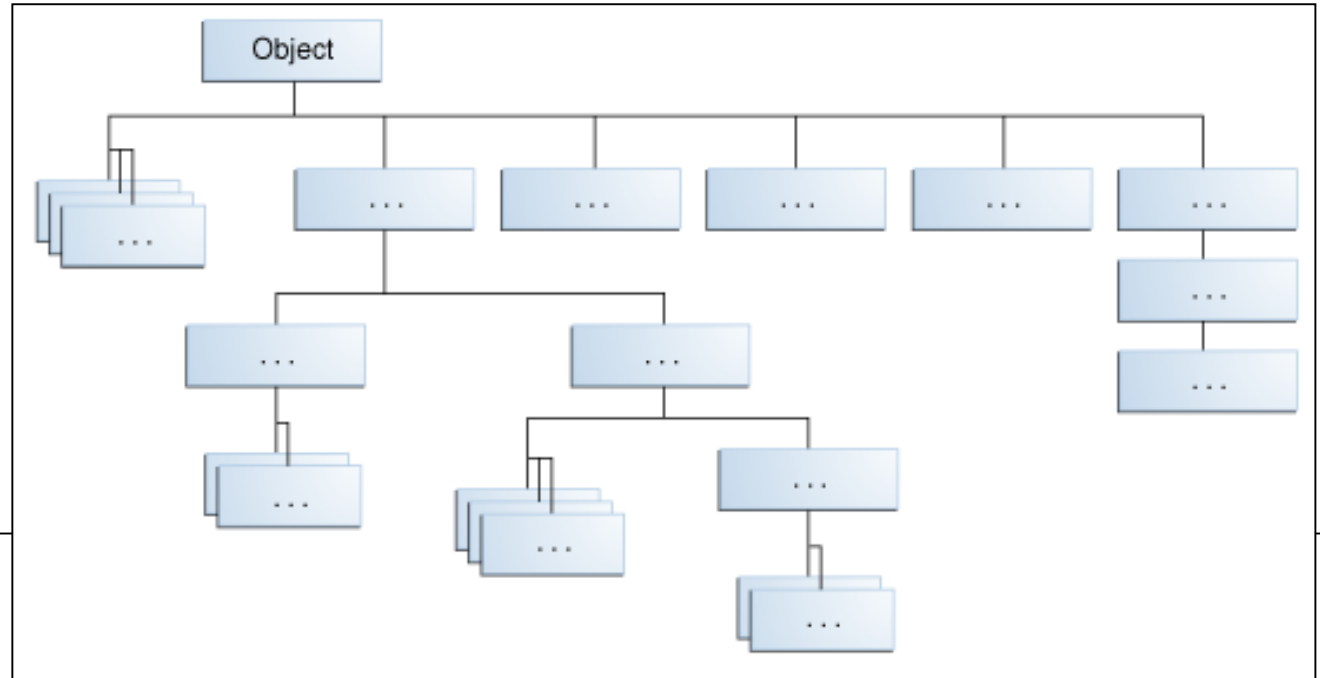
Provides classes that are fundamental to the design of the Java programming language



Special class; no need
to import it.

Class<T>	Instances of the class Class represent classes and interfaces in a running Java application.
Object	Class Object is the root of the class hierarchy.
String	The String class represents character strings.
StringBuilder	A mutable sequence of characters.
System	The System class contains several useful class fields and methods.

Object Class



java.lang

Class Object

java.lang.Object

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

Since:

JDK1.0

Object Class

Method Summary

Methods

Modifier and Type	Method and Description
protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
void	notify() Wakes up a single thread that is waiting on this object's monitor.
void	notifyAll() Wakes up all threads that are waiting on this object's monitor.
String	toString() Returns a string representation of the object.
void	wait() Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.
void	wait(long timeout) Causes the current thread to wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.
void	wait(long timeout, int nanos) Causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object, or some other thread interrupts the current thread, or a certain amount of real time has elapsed.

Object Class: equals() method


```
public class Book {  
    ...  
    @Override  
    public boolean equals(Object obj)  
    {  
        if (obj instanceof Book)  
            return ISBN.equals((Book)obj.getISBN());  
        else  
            return false;  
    }  
}
```

```
Book firstBook    = new Book("0201914670");  
Book secondBook  = new Book("0201914670");  
  
if (firstBook.equals(secondBook))  
{  
    System.out.println("objects are equal");  
}  
else  
{  
    System.out.println("objects are not equal");  
}
```

Object Class: hashCode() method

Used by collections,
primarily HashMap and
HashSet

Returns an int for
indexing ; must be
identical for objects that
are equal



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

String Class

Strings are
immutable

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

StringBuffer Class

StringBuffer is used for Strings that can change e.g. 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.

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



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

Console

Covered in this lecture:

⊕ **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



Any questions?