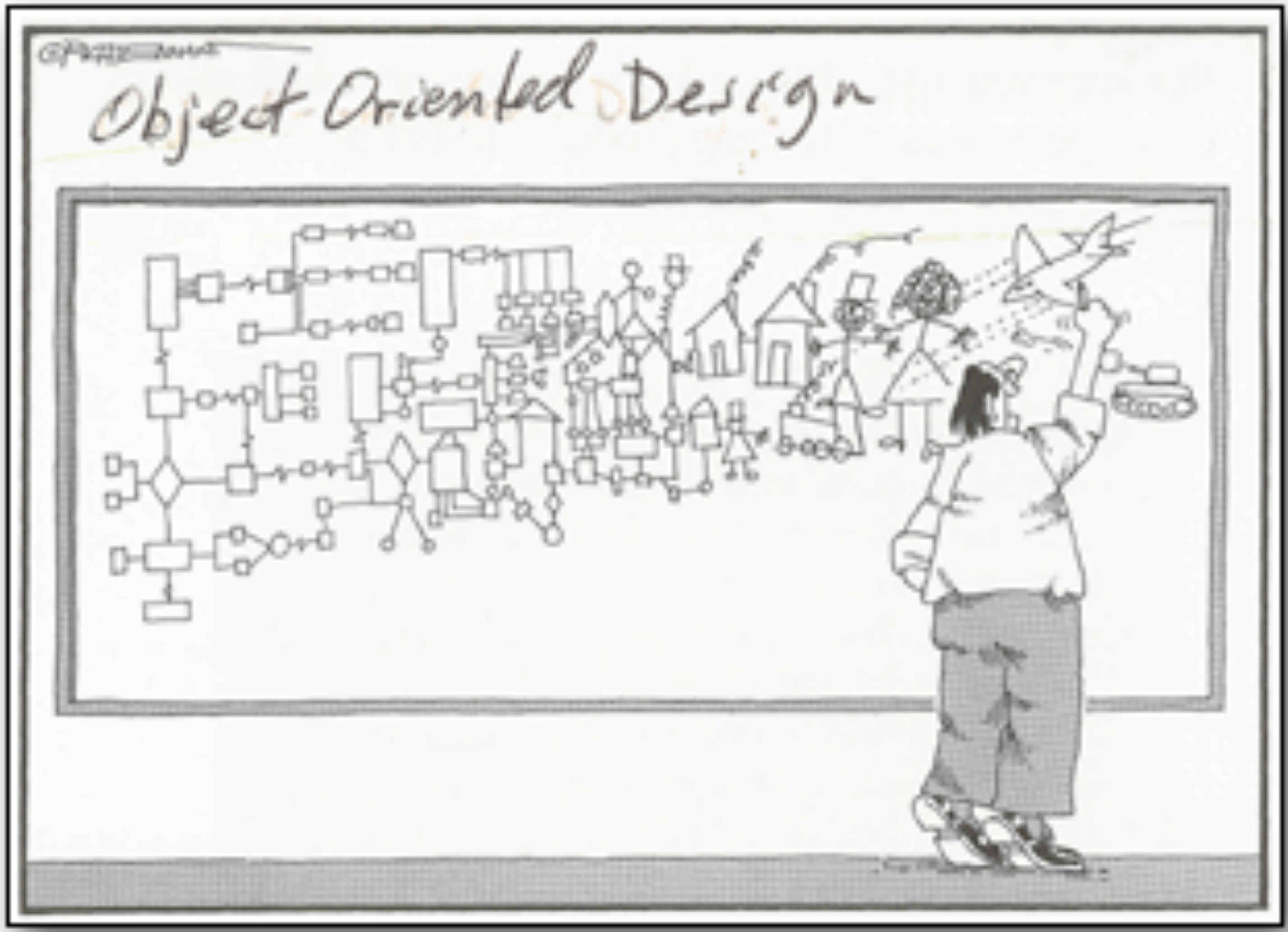
Introduction to Objects and Classes

David Greenstein
Monta Vista High School

Object Lesson



Early Programming Languages

- Instructions and data were kept separate

Instructions (Fortran)

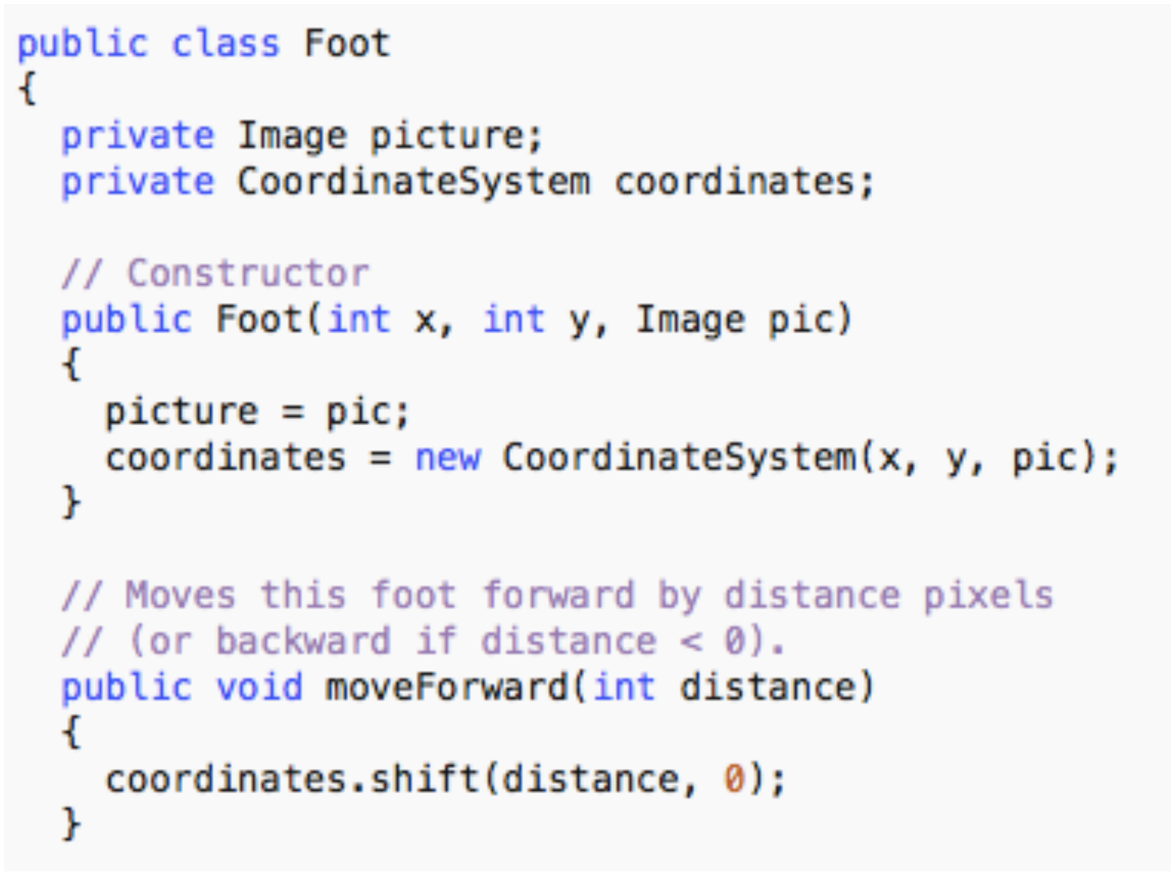
```
    DO 10, I = 1,100
         READ(3,*,END=20,ERR=900)COUNT(I),A(I),NAME(I)
         FILENO = I
   10 CONTINUE
   20 WRITE(*,*)'Input complete. Number of records: ',FILENO
…
  900 STOP 'Error in input file'
      END
```

Data input

```
23 2 ForrestGump
103 6 HanSolo
271 3 IndianaJones
…
```

Object-Oriented Languages

- Instructions and data are now bound to each other
- A paradigm that relates to the real world



```java
public class Foot
{
  private Image picture;
  private CoordinateSystem coordinates;

  // Constructor
  public Foot(int x, int y, Image pic)
  {
    picture = pic;
    coordinates = new CoordinateSystem(x, y, pic);
  }

  // Moves this foot forward by distance pixels
  // (or backward if distance < 0).
  public void moveForward(int distance)
  {
    coordinates.shift(distance, 0);
  }
```

Objects

- Objects contain values called “**fields**” that hold information about the ***state*** of the object. (Also called “attribute”.)
  - Object: Person
  - Fields: Weight, eye color, age, grade, etc.

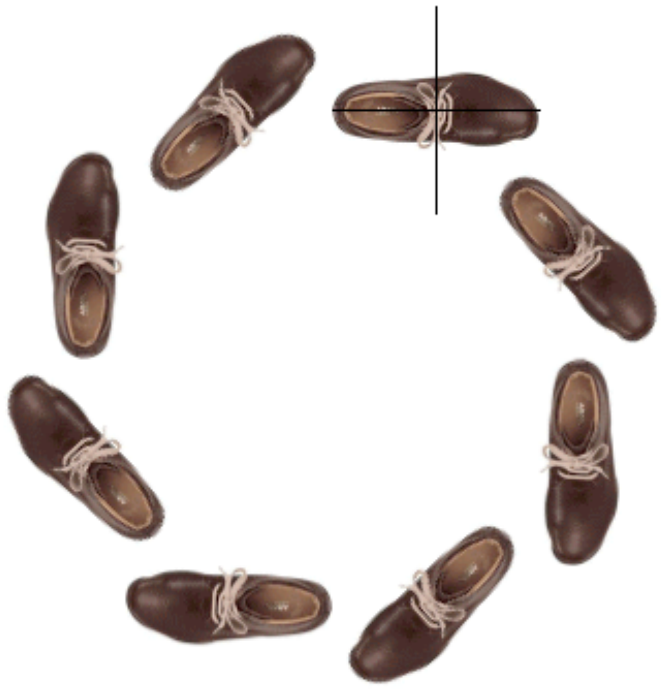
- Objects contain procedures called “**methods**” that describe the ***behavior*** of the object. Methods can also be used for getting information, often called ***message passing***.
  - Object: Person
  - Methods: Report weight, procedure getting to school, etc.

Objects (cont.)

- Can **model real-world objects** (lions, tigers, bears, oh my!)
- Can **model GUI components** (frame, panel, label, etc.)
- Can **model software entities** (events, files, images, etc.)
- Can **represent abstract concepts** (e.g. rules of a game)

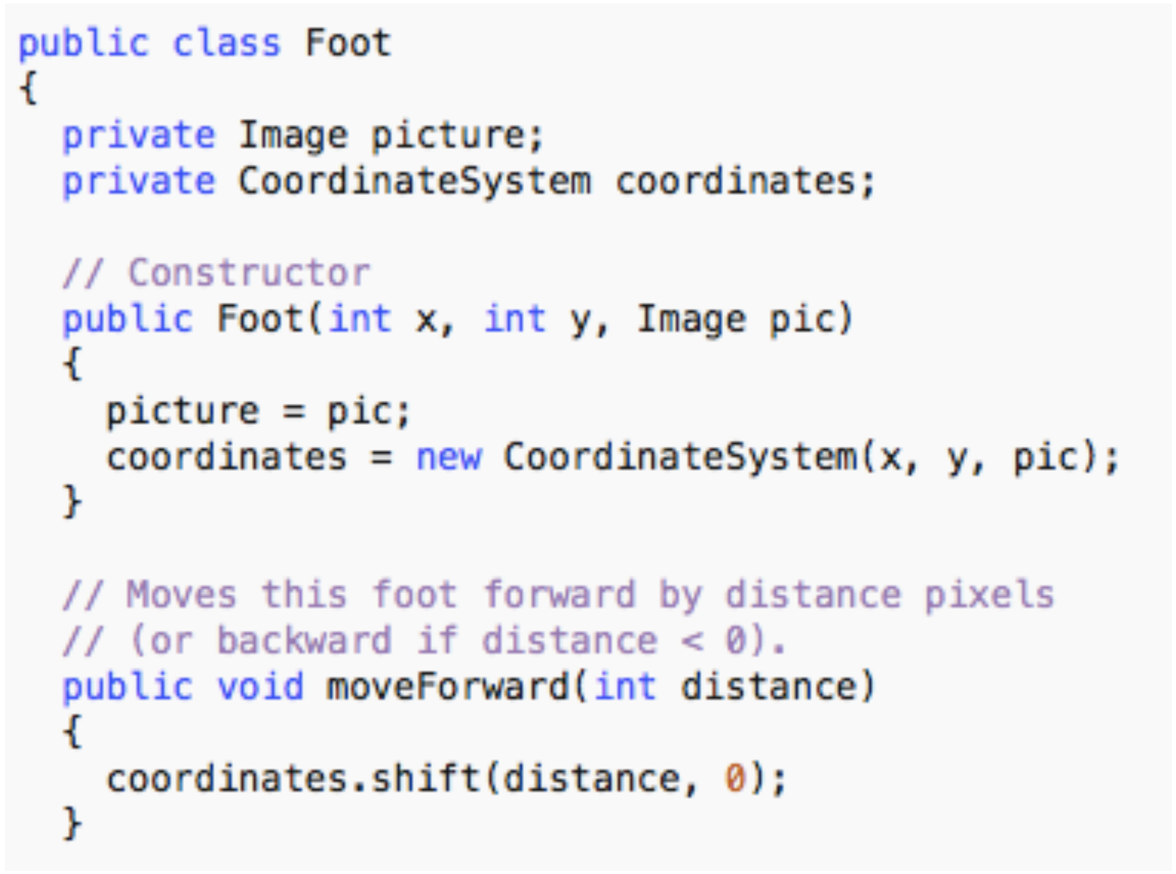
FootTest Example

```
Foot foot = new Foot(x, y, shoe);

for (int count = 1; count <= 8; count++)
{
  foot.draw(g);
  foot.turn(45);
  foot.moveForward(stepLength);
}
```



Classes and Objects

- A **class** is a description of a particular type of object, also called a ***class definition.*** This definition is the source code of the program.

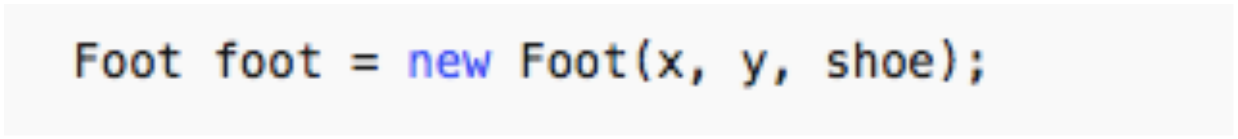


```java
public class Foot
{
  private Image picture;
  private CoordinateSystem coordinates;

  // Constructor
  public Foot(int x, int y, Image pic)
  {
    picture = pic;
    coordinates = new CoordinateSystem(x, y, pic);
  }

  // Moves this foot forward by distance pixels
  // (or backward if distance < 0).
  public void moveForward(int distance)
  {
    coordinates.shift(distance, 0);
  }
```

Classes and Objects

- A **class** is a description of a particular type of object, also called a ***class definition.*** This definition is the source code of the program.

- An **object** is called an ***instance*** of a class. A program can create more than one object (instance) of the same class.



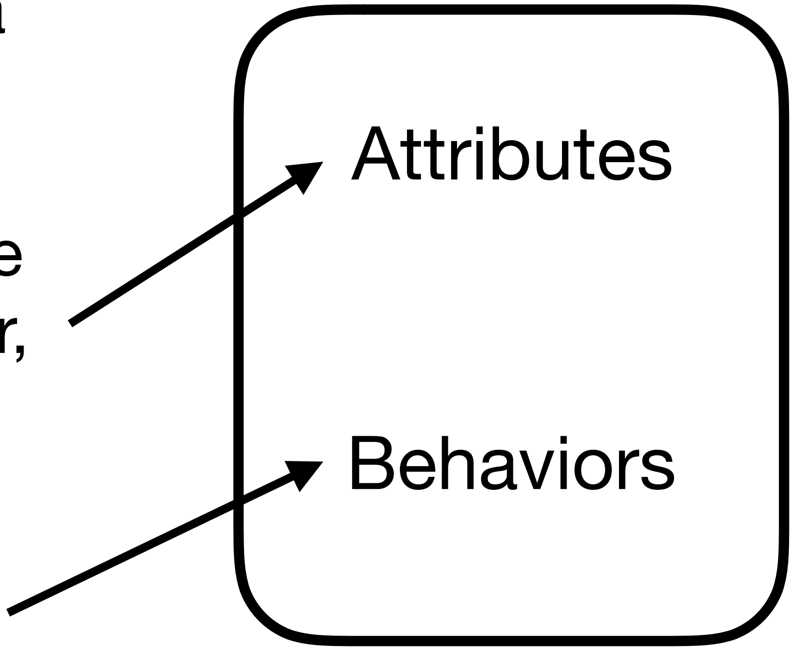
```
Foot foot = new Foot(x, y, shoe);
```

In other words …

Class

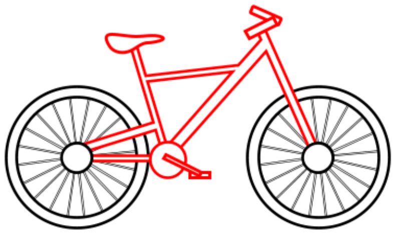
- A blueprint for objects of a particular type.
- Defines the structure of the attributes or fields (number, types).
- Defines the behaviors or methods of its objects.

Object



Example

Class: Bicycle



Attributes:

- Color
- Weight
- Type
- Material

Behaviors:

- Shift gear up
- Shift gear down
- Pedal
- Brake

$$\Big\} \longrightarrow$$

Object: My Bike

Attributes:

- Color = Silver
- Weight = 26 lbs.
- Type = Recumbent
- Material = Titanium

Behaviors:

- shiftUp()
- shiftDown()
- pedal()
- brake()

Class vs. Object

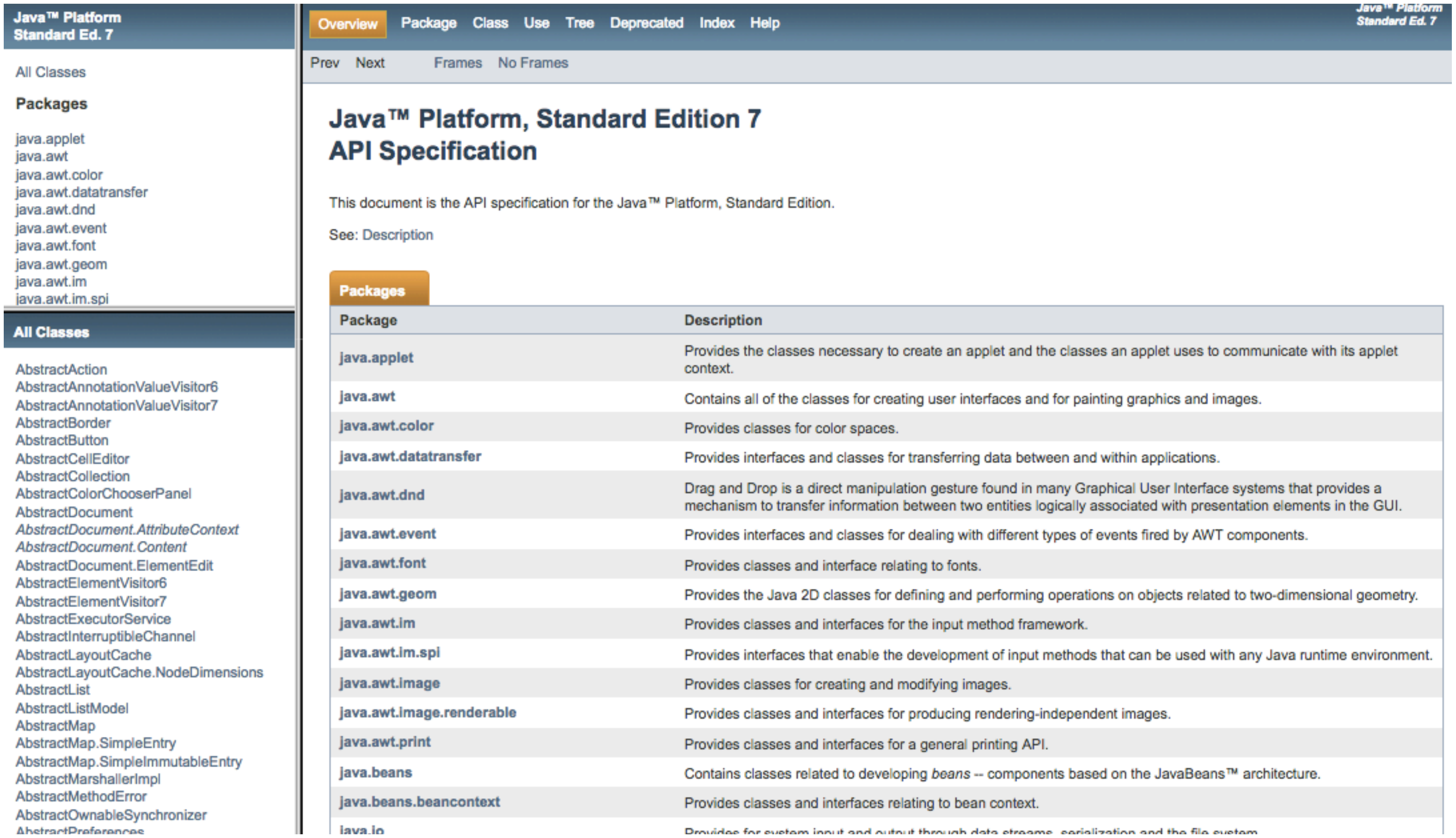
CLASS	OBJECT
A piece of source code	An entity that exists during execution
Definition stored on the hard drive	Exists in RAM
Written by the programmer	Created and destroyed by the running program

Libraries of Classes

- Java programs (class definitions) are not written from scratch.
- There are hundreds of Java classes already written and available.
- Libraries are organized into “packages”, for example:
  - java.util — miscellaneous utility classes
  - java.awt — windowing and graphics toolkit
  - javax.swing — GUI development package

Java API's

- API stands for *Application Program Interface* and contains all the documentation on Java's library of classes.



Java™ Platform
Standard Ed. 7

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi

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

Java™ Platform, Standard Edition 7
API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Packages

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 *beans* -- 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.

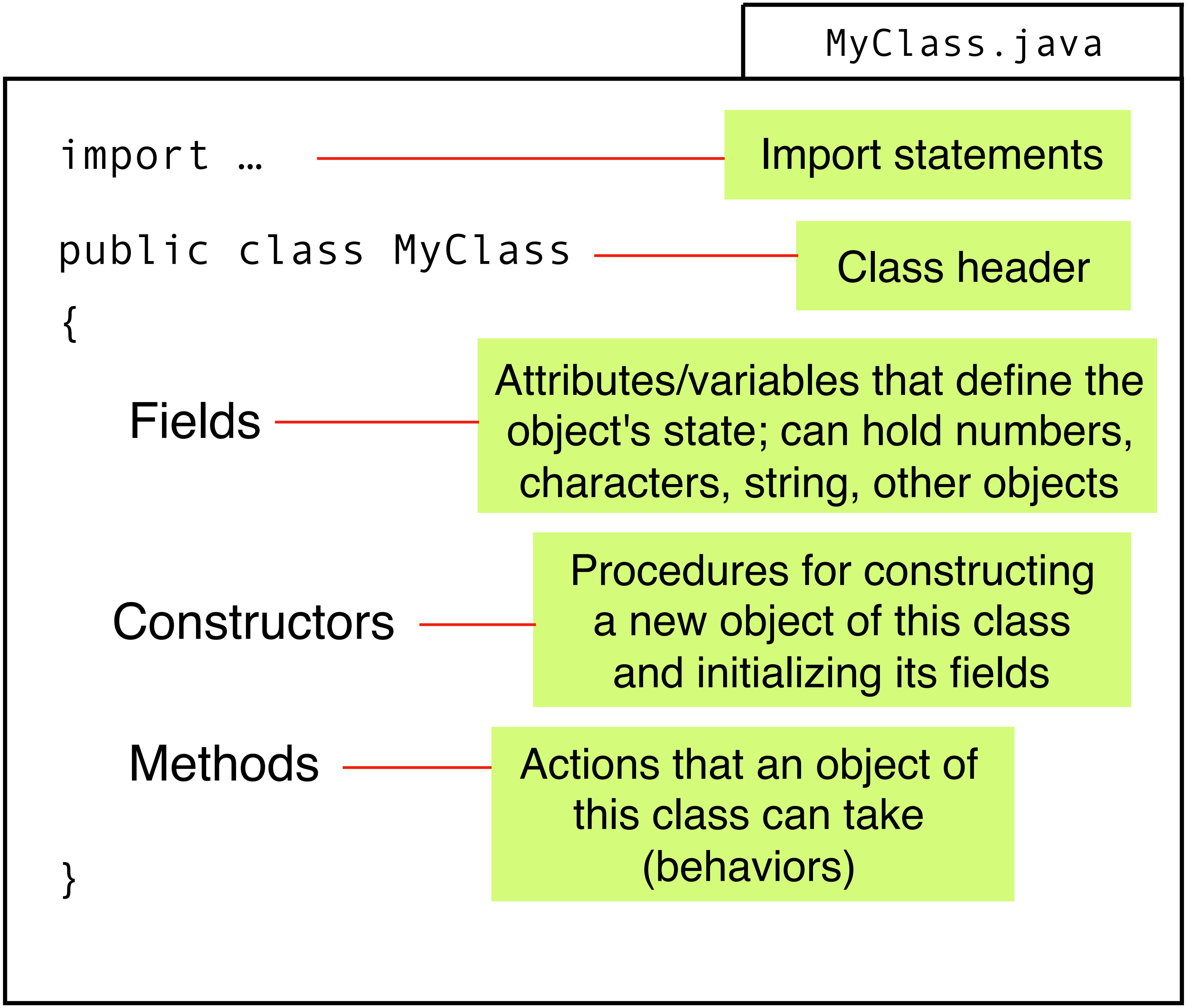
Importing Classes

- The method for getting access to these classes is to **import** them into your source code.

```
import java.awt.Color;
import java.util.ArrayList;
import javax.swing.JPanel;
```

- There are so many library classes from which to choose! Learning about these classes, where to find them, and how to use them *takes time* and *lots of practice*.
- Luckily, you get the **java.lang** package imported automatically. It contains important classes like System, Math, Object, and String.

A Short Lesson on Creating a Class Description



```
import …

public class MyClass
{
    Fields

    Constructors

    Methods

}
```

Import statements

Class header

Fields: Attributes/variables that define the object's state; can hold numbers, characters, string, other objects

Constructors: Procedures for constructing a new object of this class and initializing its fields

Methods: Actions that an object of this class can take (behaviors)

Fields

```
private [static] [final]  datatype  name;
```

Usually private

Optional:
means the field is shared by **all** objects from the class

Optional:
means the field is a **constant**

Primitive: `int`, `double`, etc.
or an
Object: `String`, `Color`, etc.

You decide

```
private double weight;
private String type;
private final double LBS_PER_KG = 2.2046;
```

Constructors

```
public class Coordinate {
   private int x, y;

   . . .

   public Coordinate() {
      x = 0;
      y = 0;
   }

   public Coordinate(int myX, int myY) {
      x = myX;
      y = myY;
   }

   . . .

}
```

Class and Constructor names must be the same

There can be more than one constructor defined

A Constructor with no arguments is called a **no-args constructor**

Methods

```
public class Coordinate {
   private int x, y;
   . . .

   public void addToX(int num) {
      x += num;
   }

   public double distance(Coordinate other) {
      return Math.sqrt( Math.pow(x - other.x, 2)
                      + Math.pow(y - other.y, 2));
```

Values passed to the method are called **parameters**

Methods can **return** values or objects (message passing)

A **void** return type means the method only does an action internal to the object

```
Coordinate origin = new Coordinate();
Coordinate point = new Coordinate(4, 3);
two.addToX(5);
double dist = point.distance(origin);
```

Methods can declare and use variables called **local variables**

OOP Advanced Features

- Encapsulation and Information Hiding
  - A class interacts with other classes only through constructors and public methods.
  - Other classes do not need to know the mechanics (implementation details) of a class to use it effectively.
  - Encapsulation facilitates team work and program maintenance (making changes to the code).

- Inheritance
  - The programmer can create a new class that extends an existing class, and it is called a **subclass**.
  - The class extended by the subclass is called the **superclass**.
  - All classes have a superclass called **Object** and it can be found in the **java.lang** package.

The features will be discussed more in Chapters 9 and 11

Questions?