



Introduction to Objects and Classes

David Greenstein
Monta Vista High School

Early Programming Languages

1 of 18

- 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

2 of 18

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

```
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

3 of 18

- 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.)

4 of 18

- 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

5 of 18

```
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

6 of 18

- 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.

```
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

7 of 18

- 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);
```

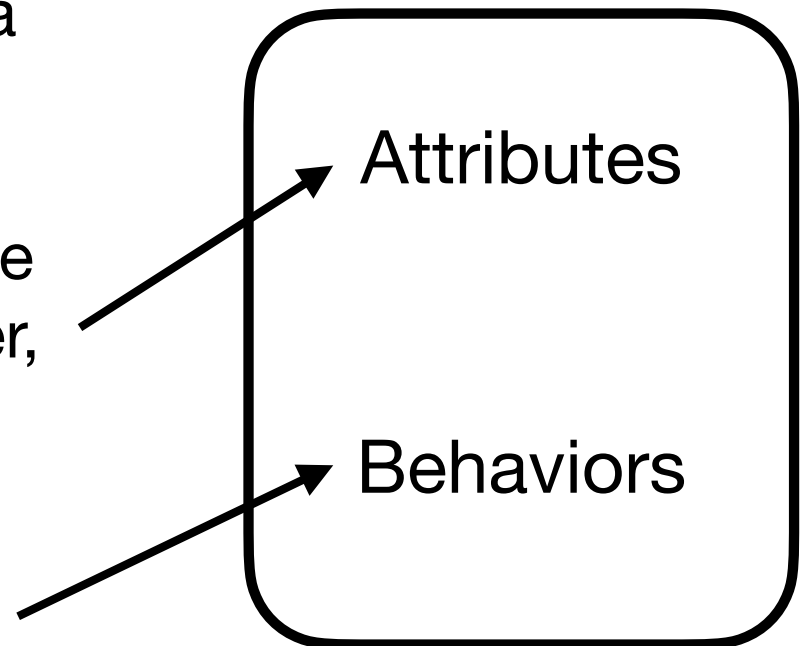
In other words ...

8 of 18

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

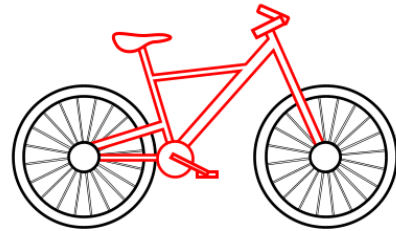


Example

9 of 18

Class: Bicycle

Object: My Bike

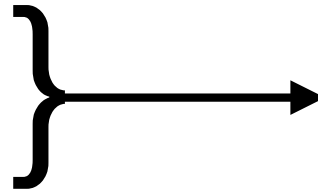


Attributes:

Color
Weight
Type
Material

Behaviors:

Shift gear up
Shift gear down
Pedal
Brake



Attributes:

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

Behaviors:

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



Class vs. Object

10 of 18

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

11 of 18

- 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

12 of 18

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

OVERVIEW MODULE PACKAGE CLASS USE TREE PREVIEW NEW DEPRECATED INDEX HELP Java SE 17 & JDK 17

SEARCH:

Java® Platform, Standard Edition & Java Development Kit Version 17 API Specification

This document is divided into two sections:

Java SE

The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with `java`.

JDK

The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with `jdk`.

All Modules	Java SE	JDK	Other Modules
-------------	---------	-----	---------------

Module	Description
java.base	Defines the foundational APIs of the Java SE Platform.
java.compiler	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
java.datatransfer	Defines the API for transferring data between and within applications.
java.desktop	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
java.instrument	Defines services that allow agents to instrument programs running on the JVM.
java.logging	Defines the Java Logging API.
java.management	Defines the Java Management Extensions (JMX) API.
java.management.rmi	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
java.naming	Defines the Java Naming and Directory Interface (JNDI) API.
java.net.http	Defines the HTTP Client and WebSocket APIs.
java.prefs	Defines the Preferences API.

Importing Classes

13 of 18

- 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 ...
```

Import statements

```
public class MyClass
```

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

15 of 18

```
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**

You decide

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

```
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

17 of 18

```
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** (message passing)

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

18 of 18

- 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