Java classes, part 1

Prof Bill, Jan 2020

These are supplemental notes on:

- ❖ Java. (duh) Mostly, classes in Java
- ❖ ADT - Abstract Data Types
- ❖ OOP - Object-oriented Programming

Abstract data type (ADT) - a data type whose internal representation is hidden from the client.

Object-oriented programming (OOP) - a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).

The goal of OOP = manage complexity to increase productivity of software designers

Our "textbook" links:

- ➢ Oracle Java, docs.oracle.com/javase/tutorial/java
- ➢ Sedgewick Java, introcs.cs.princeton.edu/java/home
- ➢ Sedgewick, Algorithms, algs4.cs.princeton.edu/home

Thank you - I bow down to Noctrl's own Dr. Godfrey Muganda for his excellent Java textbook. I have liberally borrowed ideas from this book.

media.pearsoncmg.com/bc/abp/cs-resources/products/product.html#product,isbn=0134038177

Objects and classes

A **class** defines the variables and methods of objects. They are object templates.

An object's type is defined by its class.

Objects store data as **instance variables**...also called attributes or fields.

Objects perform operations via **methods**.

Important: instance variables are available to any methods in the class.

Different from primitives, you need to explicitly create objects using **new**.

```
int test7; // integer created

Random rand; // just a reference or pointer, no object yet!
rand = new Random(); // now we have one
```

The Java API has classes in its standard library, for example: Scanner, Random, PrintWriter, etc. There's a lot there, docs.oracle.com/javase/8/docs/api.

When you use a method in the Java API, you'll often need to import the package where this code resides.

```
import java.util.Scanner;
import java.util.Random;
import java.io.PrintWriter;
```

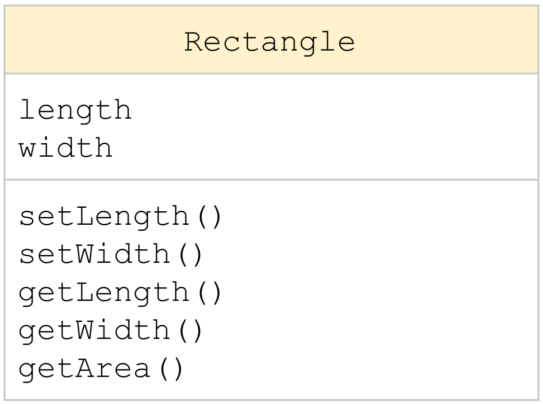
How do I know what to import to use the Java API? Example: google java PrintWriter...and look at the class definition.

The java.lang package contains the String class and is automatically imported by Java.

Writing a simple class

Define your own class means (pretty much) defining the variables and methods.

This is a **UML Diagram**. It’s just: 1) class name, 2) variables, and 3) methods.



/ You can express private/public access and data types in UML diagram if you want */*

You can code the Rectangle class from this diagram:

- ➢ private variables, public methods
- ➢ data type of length, width is assumed to be int (or double, you decide)
- ➢ write method stubs (signature), leave guts for later
- ➢ it should compile!
- ➢ then start coding ‘er up, one method at a time, add ctor

Definitions:

- **Accessor** or “get” methods - returns the value of a field, example: getWidth()
- **Mutator** or “set” methods - set or define value for a field, example: setLength()
- The goal is **data hiding**. This is an object-oriented programming (OOP) concept that hides the complexity of how data is stored in an object. Interaction with objects is usually via methods. Safety, too!

Constructors (ctors)

A **constructor** is a method called to create an object; **ctor** (see-tor) is a cooler nickname. We usually call the ctor with the **new** keyword.

```
Rectangle r = new Rectangle( 5, 10);
```

A ctor with no parameters is called the **default constructor**.

```
Rectangle r2 = new Rectangle();
```

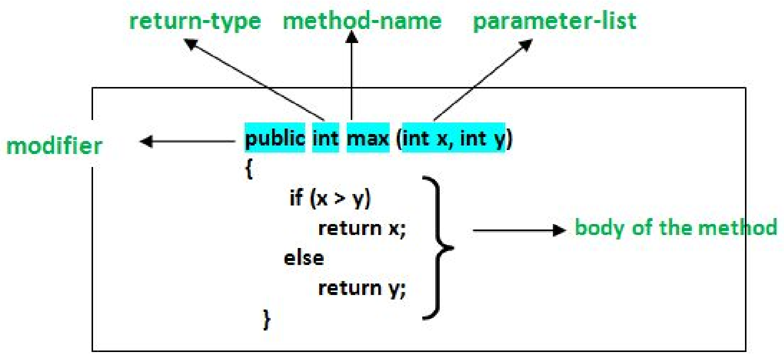
The ctors are defined in the class. These methods are special and always have the same name as the class.

```
public class Rectangle {
    // lots of stuff...

    // default ctor
    public Rectangle() {
        // insert code here
    }

    public Rectangle( int len, int width) {
        // insert code here
    }
}
```

method signature - the declaration of a method: name, return value, and parameters



```
public int max (int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
```

Source: www.geeksforgeeks.org/methods-in-java/

overload - when two methods (ctor or not) have the same name, but different signature; Java picks between overloaded methods by looking at the method arguments

override - when a subclass has the same method signature of a parent class (more later when we cover inheritance, etc)

An object variable is a **reference**, or **pointer**.

```
Rectangle test1 = new Rectangle(); // new object
Rectangle test2 = test1; // not a copy, just a pointer
Rectangle test3; // no object, just null (zero) pointer
```

Interfaces

An **interface** specifies only the behavior of a class. It is a completely abstract class and does not contain any real code.

An interface is a **contract**. Any class that **implements** an interface guarantees that the methods in the interface will be available.

Example:

```
public interface Animal {
  public void animalSound(); // no body to these methods!
  public void run();
}

public class Squirrel implements Animal {
  public void animalSound() {
    //insert code here
  }
  public void run() {
    // insert code here
  }
}
```

Interfaces are a powerful concept in Java:

- ➔ Specify the method signatures of a class, without all the code
- ➔ Interfaces still compile!
- ➔ Allows for inheritance and polymorphism between classes

Next up: Java inheritance, polymorphism, abstract classes, etc.

Coming soon... in Java classes, part 2!