### **Reading**

Please read Chapter 9 of book.

### **Objects**

Objects represent things in the real world. E.g. person, circle, bike, tic tac toe grid, loan, movie, home rental, online order, etc.

Objects are made up of (1) state and (2) behavior.

**State** of an object are its attributes (really the member variables belonging to the object).

**Behavior** of an object is the set of actions the object can perform (really the methods belonging to the object).

### **Class vs. Object**

**Class** is a template or blueprint.

**Object** is an instance of the Class.

E.g. Circle is a class, whereas there may be two circle Objects (one with radius 2 and one with radius 100).

**Instantiation** is act of creating an Object.

Each object is called an **Instance** of the class (eg. the two instances above of the Circle class)

**Constructors** are special methods of a class used to create an object.

### **Example Class**

Consider the following class definition for a circle.

class Circle {

// data field or attribute

double radius;

// Constructors

Circle() {

radius = 1;

}

Circle(double r) {

radius = r;

}

// Methods

double getArea() {

return radius \* radius \* Math.PI;

}

double getPerimeter() {

return 2 \* radius \* Math.PI;

}

void setRadius(double r) {

radius = r;

}

}

Note that this class does not have a main method, so it can not be run by itself.

We will refer to a class with a main method as the **main class**. We must add a main class to the above to make it runnable.

### **UML Diagrams**

Unified Modeling Language (UML) notation is a standard way of representing Class hierarchies and interactions.

See how class names, variable and methods are represented below.

CircleUML.png

Note that:

* Class name is at the top
* Data field names and types are in the middle
* Constructors and methods are in the bottom.

For now, we are not showing visibility modifiers (public, private, etc.) here; they are covered later.

### **Constructors**

Special kind of method:

* Must have same name as the class
* Do not have a return type
* Invoke by using the new operator

Can be overloaded like other methods.

If no constructor defined, a default constructer (public, no arguments, empty body) is automatically created.

### **Using a Class**

To use the above class, you will need to add a main class (called MyProgram below) that utilizes the Circle class.

public class MyProgram {

public static void main(Strings[] args) {

Circle circle1 = new Circle(25);

Circle circle2 = new Circle();

Circle circle3 = new Circle(100);

}

}

ObjectsUML.png

Variables circle1, circle2, and circle3 are **reference variables**.

You can put both the Circle and MyProgram classes in the same file called MyProgram.java. Only one class in a file can be public; the public class has the same name as the file.

Note that the compiler will produce one .class file for each class. So, in this case, it will produce Circle.class and MyProgram.class.

### **Access Objects Data and Methods**

An objects data and methods can be accessed via the dot (.) notation.

circle1.radius references the radius in circle1.

circle1.getArea() invokes the getArea method of circle1.

Also note that objects, like circle1, are passed into methods by reference.

Recall the difference between **primitive types** (int, boolean, double, etc.) and **reference types** (arrays, object, etc.). A variable of a primitive type holds a value of that type. A variable of a reference type holds the location where the object is located in memory.

Aside: Java does provide object versions of primitive datatypes, eg. Boolean, Character, Double, Integer, etc.

### **Static Keyword**

A variable like radius above is known an **instance variable**. This means each object (circle1, circle2, etc.) has its own copy of the variable.

By contrast, a **static variable** (declared as below) is shared by all instances of the class; these sometimes also called class variables.

static int circleCount = 0;

If one object modifies a static variables, all objects see the change.

StaticDataUML.png

A **static method** can also be defined similarly.

Question: Where have we seen a static method before? Hint: Look back at the simple Hello World example.

Static method can be called without needing to create an object.

Also note that static methods can not refer to any instance variables.

How can we use a static variable to count how many objects of a particular type were created?

class Circle {

// data field or attribute

double radius;

// number of objects created

**static int circleCount = 0;**

// Constructors

Circle() {

radius = 1;

**circleCount++;**

}

Circle(double r) {

radius = r;

**circleCount++;**

}

// Methods

double getArea() {

return radius \* radius \* Math.PI;

}

double getPerimeter() {

return 2 \* radius \* Math.PI;

}

void setRadius(double r) {

radius = r;

**static int getNumberOfCircles() {**

**return circleCount;**

**}**

}

All the following are ok to use.

circle1.circleCount

Circle.circleCount // better as it shows that using static variable

circle1.getNumberOfCircles()

Circle.getNumberOfCircles() // better as it shows that using static method

The updated UML diagram with static variables and static methods is below. Note that static variable and methods are underlined.

CircleStaticUML.png

### **Visibility Modifiers**

We can control the visibility of classes, methods, and variables by using the keywords such as public, private, etc.

If using **public** for classes, methods and variables, they are accessible by anyone.

If no visibility modifier used (like for Circle above), then they are accessible by any class in the same [package](https://www.w3schools.com/java/java_packages.asp)

[(Links to an external site.)](https://www.w3schools.com/java/java_packages.asp)

.

If using **private** for methods and variables, they are accessible within the same class.

To protect data and make the class easier to maintain, make the variables private. Provide public getter (accessor) and setter (mutator) methods to manage the data. This is known as **data encapsulation**.

**public** class Circle {

// data field or attribute

**private** double radius;

// number of objects created

**private** static int circleCount = 0;

// Constructors

**public** Circle() {

radius = 1;

circleCount++;

}

**public** Circle(double r) {

radius = r;

circleCount++;

}

// Methods

**public double getRadius() {**

**return radius;**

**}**

**public** double getArea() {

return radius \* radius \* Math.PI;

}

**public** double getPerimeter() {

return 2 \* radius \* Math.PI;

}

**public void setRadius(double r) {**

**radius = r;**

**public** static int getNumberOfCircles() {

return circleCount;

}

}

The updated UML diagram with visibility modifiers will look like the below. Note the "-" for private and "+" for public.

CircleEncapsulationUML.png

### **Using Objects**

An object is passed by reference to a method.

Can declare arrays of objects.

Class variables and methods are available for use (scope) in the entire class regardless of where they appear.

An **immutable** object is one that can't be changed once created. You can define classes to support this behavior. [String](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html)

[(Links to an external site.)](https://docs.oracle.com/javase/7/docs/api/java/lang/String.html)

is an example of an immutable object.

### **Keyword "this"**

The keyword **this** can used object to refer to itself.

It can be used (as in the example below)

* in a constructor to call another constructer
* to refer to variables that have become hidden

class Circle {

// data field or attribute

double radius;

// Constructors

Circle() {

**this(1.0);**

}

Circle(double **radius**) {

**this.**radius = **radius**;

}

...

}