The Software Object

Lesson 1.2

Learning Outcomes

LO 1.2.1	**Name** classes appropriately
LO 1.2.2	**Assert** necessary attributes for a class definition
LO 1.2.3	**Implement** indispensable methods for a class definition
LO 1.2.4	**Identify** and **implement** different types of constructors for a class definition

Learning Objectives

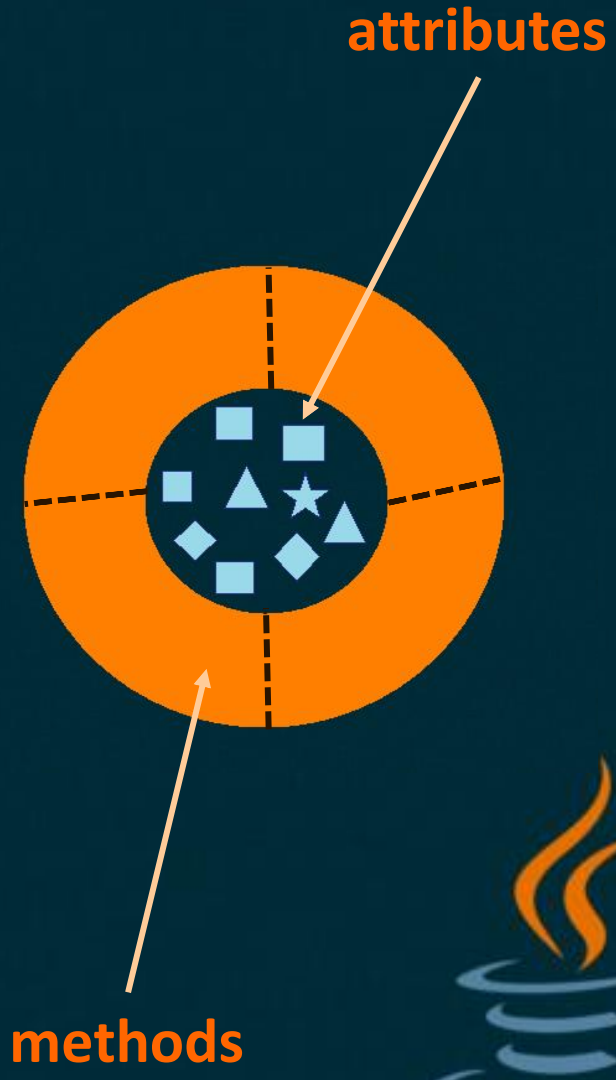
LO 1.2.5	**Instantiate** objects from classes
LO 1.2.6	**Call** methods from instantiated objects
LO 1.2.7	**Differentiate** object identity from object equality

Software Object

A **software object** maintains its *state* in **attributes** and implements its *behavior* with **methods**.

These variables and methods are formally known as **instance attributes** and **instance methods**.

An **instance** of an object is its *identity*.



How do we model software objects in Java?

CLASSES

Class

Four ways you could think of a class

- A **factory** manufactures objects according to some *blueprint*.
- A **set** specifies what features its member objects will have.
- A **template** allows us to produce any number of objects of a given shape.
- A **dictionary** definition describes an object as precisely as possible.

Class Name

The **class name** must represent the question: **What am I?**

This is not to be confused with IDENTITY (Who am I?) and must be *unique*.

In Java *standards*, class names should start with an uppercase letter and followed by lowercase letters and/or digits. If the name consists of multiple words merged together, the first letter of each word should be an uppercase letter.

Example: `HelloWorldProgram, R2D2, Person`

LO 1.2.1 Name classes appropriately

Create 2 classes by selecting any objects you see around you and name them appropriately.

Class Attributes

The **class attributes** are variables that contain the current state of the object. These are the properties of a class and represents the question: **What do I have?**

Class Attributes

In *Java*, attributes are implemented as variables.

A basic syntax of an attribute in Java:

`<access_modifier> <data_type> <variable_name>`

Example:

```
    public int age;
    public String[] names;
    public double total;
```

this Keyword

In Java *standards*, **this** keyword is used to call any attribute or method that is owned by the class itself. It is only optional with respect to compilation but is usually used to differentiate local/global/static variables or functions from the class attributes/methods.

Example:

```
    this.age;
    this.names;
    this.total;
```

LO 1.2.2 Assert necessary attributes for a class definition

Assign at least 2 attributes to each of your created classes based on their possible states.

Class Methods

The **class methods** represents the behavior of the object. In Java, methods are implemented as **functions**.

A basic syntax of a function in Java:

```
<acc_mod> <ret_type> <func_name>(<params>){
      <implementation_body>
}
```

Example:

```
public int add(int x, int y){
  return x+y;
}
```

LO 1.2.3 Implement indispensable methods for a class definition

Implement at least 2 methods appropriate for each of your created classes based on their behavior.

Constructors

A **constructor** is a method that is invoked to create an object.

It is normally used to initialize data members. It has the same name as the name of the class. It can receive parameters but it has no return value.

In Java, there is always **at least one constructor** in every class. You can create as many type of constructors as you want in a class as long as the **parameter types and order** are **distinct**.

Constructors

A **default constructor** is a constructor which does not require any parameter when creating a new object.

Example:

```
public Person()
{
  this.age = 0;
}
```

```
public Animal()
{
   this.numberOfLegs = 2;
}
```

Constructors

A **copy constructor** is a constructor that accepts a parameter of the same type as the class wherein its role is to create an object which contains the same state/conditions as the object parameter.

Example:

```
public Person(Person p)
{
    this.age =
        p.age;
}
```

```
public Hat(Hat h)
{
    this.color =
        a.color;
}
```

LO 1.2.4 Identify and implement different types of constructors for a class definition

Implement a default and copy constructors appropriate for each of your created classes. Include also one constructor that is neither of the two.

Instantiating Objects

You can **create/instantiate** objects by calling the constructor method of a class.

Example:

```
Person p = new Person(); //default constructor

p = new Person(6); //constructor

Animal a1 = new Animal(1, true); //constructor

Animal a2 = new Animal(a1); //copy constructor
```

LO 1.2.5 Instantiate objects from classes

Now, instantiate 2 different objects from each of your created classes.

Calling Methods from Objects

You can **call** methods from objects during *runtime* as long as the **object** is **not null**.

Example:

```
Person p = new Person();
p.walk(); //does not cause an error if walk is
              implemented in class Person with
              the default signature

p = null;
p.walk(); //causes an error because the
              identifier p is null
```

LO 1.2.6 Call methods from instantiated objects

Then, call the methods from each of your instantiated objects.

Object Identity vs. Equality

Different objects must have different identities. Different objects may have exactly the same state.

`if(obj1 == obj2)` tests **IDENTITY**

`if(obj1.equals(obj2))` tests **EQUALITY**

LO 1.2.7 Differentiate object identity from object equality

Demonstrate how you can establish comparison on same and different object identities with the current objects you have.

Also, with the same objects, demonstrate how you can establish comparison on equal and unequal objects.