Object Oriented Programming Methodology

Module 2 : Class, Object, Method and Constructor

Faculty In-charge:
Pragya Gupta
E-mail: pragya.g@somaiya.edu

Contents

- Class
- Objects
- Methods
- Constructors

Parts of a class

- The class contains two different sections:
  - Variable declarations: Describe state
  - Method declaration: Describe behavior

```
classDeclaration {
memberVariableDeclarations
methodDeclarations
}
```

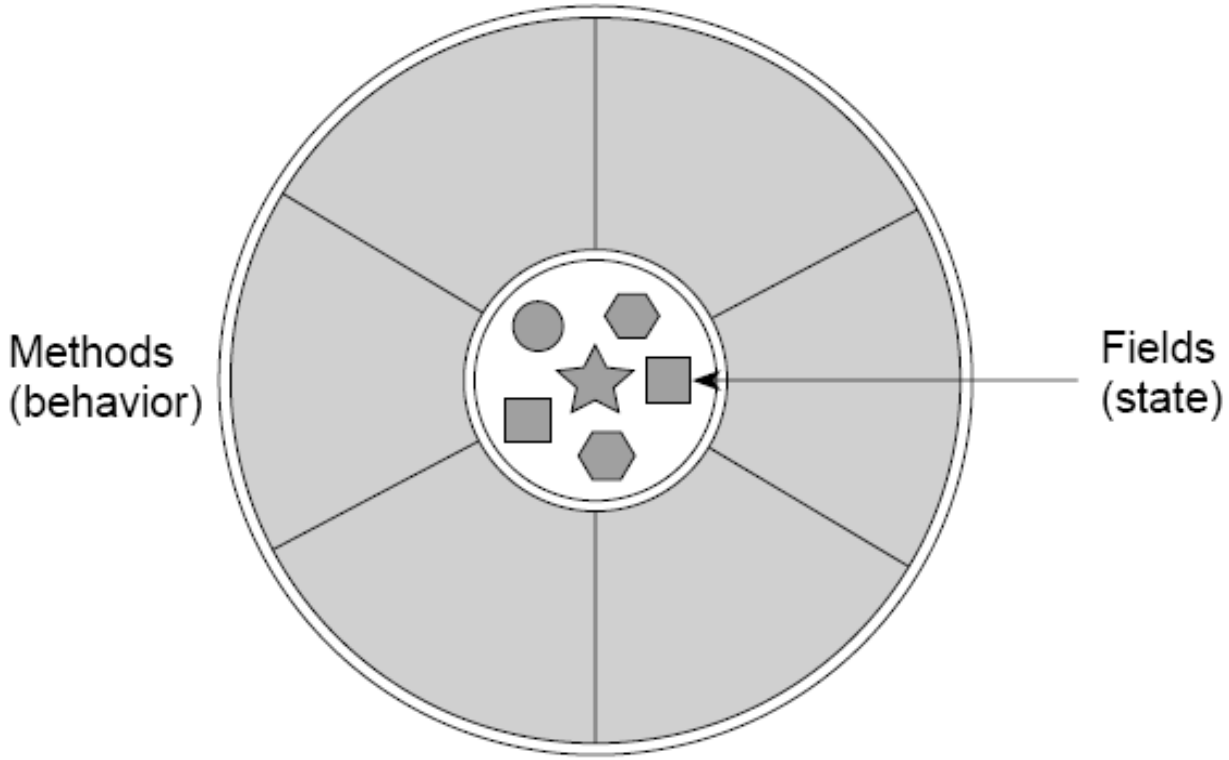
Example

Class SalesTaxCalculator for calculating and displaying the tax

```
class SalesTaxCalculator {
        float amount=100.0f;
        float taxRate=10.2f;
        void calculateTax() {
                float taxAmt = amount*taxRate/100;
                System.out.println(taxAmt);
        }
}
```

Why should we use classes and objects?

Modularity and information hiding i.e. data encapsulation can be incorporated using an object, in software. Classes, being blueprints, provide the benefit of reusability.



Creating Objects

- Java object is created with a statement like this one:

```
SalesTaxCalculator obj1 = new SalesTaxCalculator ( );
```

- This statement creates a new SalesTaxCalculator object
- This single statement *declares*, *instantiates*, and *initializes* the object

Declaring an Object

- Object declaration is same as variable declaration
  - for e.g. SalesTaxCalculator obj1;
- Generally the declaration is as follows:

  type name;
  - Where *type* is the type of the object (i.e. class name)
  - *name* is the name of the reference variable used to refer the object
- Difference between variables and objects:
  - A variable holds a single type of literal
  - An object is an instance of a class with a set of instance variables and methods which performs certain tasks depending on what methods have been defined for

Initializing an object

- By initializing an object we mean that the instances variables are assigned some values
- This task is accomplished using a constructor.
- The final object creation can be said as complete when the objects are initialized, either with an implicit constructor or an explicit constructor
- This object creation can be used in programming code:
  - SalesTaxCalculator obj1 = new SalesTaxCalculator ( );
- Here all the three operations, object declaration, object instantiation and object initialization are done by one statement only

Initializing an object

The above process actually takes place in following way:

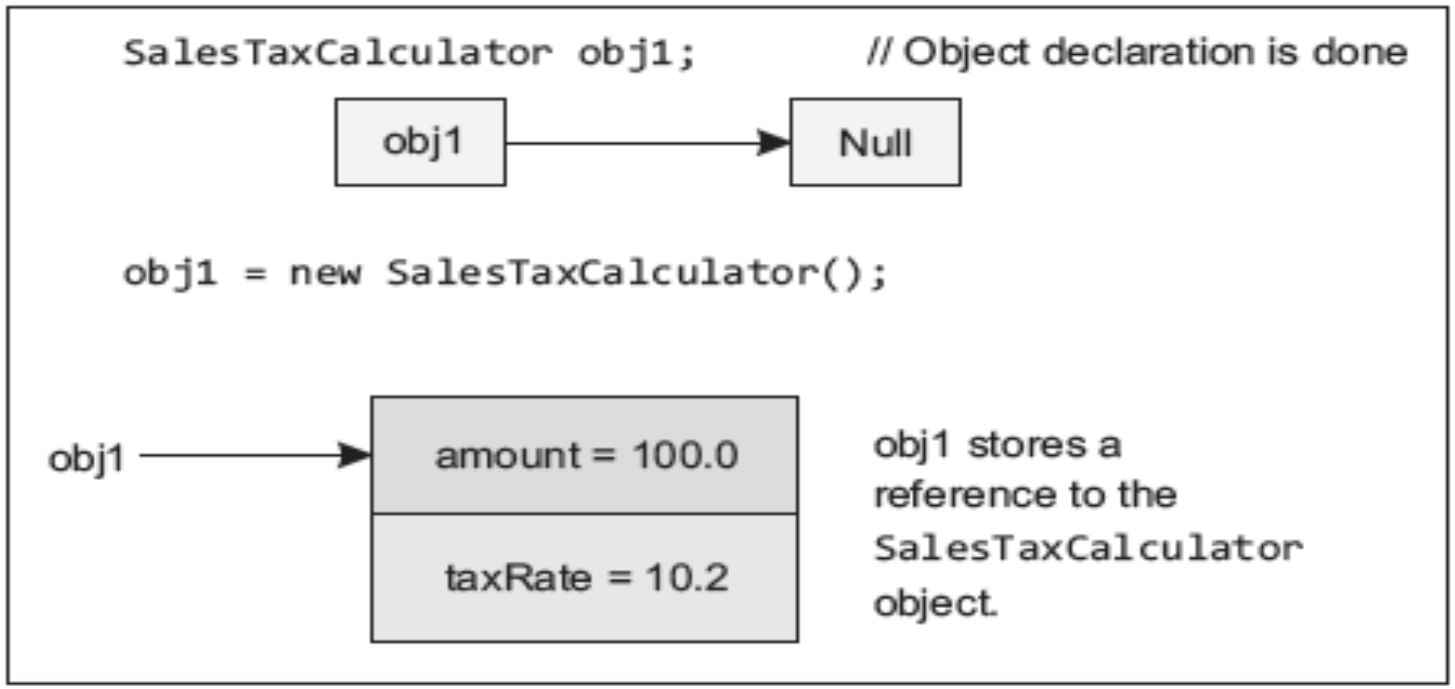


Fig. 4.3 Steps in Object Creation

Instance Variable

- A class can have many instances, each instance having its own set of variables
  E.g.

```
class SalesTaxCalculator {
        float amount=100.0f; // instance variable
        float taxRate=10.2f; //instance variable
        void calculateTax() {
        float taxAmt = amount*taxRate/100;
        System.out.println(taxAmt);   }
        public static void main (String args[ ])          {
        SalesTaxCalculator obj1 = new SalesTaxCalculator();
        SalesTaxCalculator obj2 = new SalesTaxCalculator();
System.out.println("Amount in Object 1: "+ obj1.amount);
System.out.println("Tax Rate in Object 1: "+ obj1.taxRate);
System.out.println("Amount in Object 2: "+ obj2.amount);
System.out.println("Tax Rate in Object 2: "+ obj2.taxRate);
}}
```

Accessing Instance Variables

- For accessing value of an object
  - Objectname.variablename
- To assign values to the variables of an object
  - Objectname.variablename = value
- There are three ways of assigning values to the instance variables in the objects:
  - Assigning values directly to the instance variables as shown below,

    float amount=100.0f;
  - Assigning values through a setter method
  - Values can be assigned using constructors

Instance variable (contd.)

- Each variable declared inside a class and outside the methods
- **Except static variables,** because they are created whenever an instance (object) of the class is created
- These variables are initialized by the constructors
- In the above example the two objects obj1 and obj2 will have their own set of instance variables.
- i.e. *obj1* will have its own *amount* and *taxRate* whereas *obj2* will have its own set of *amount* and *taxRate*

Methods

- Similar to a function in any other programming languages.
- None of the methods can be declared outside the class.
- Why use methods?
  - To make code reusable
  - To parameterize code
  - For top-down programming
  - To simplify code

Method (contd.)

- General syntax for a method declaration:

  [modifiers] return_type method_name (parameter_list) [throws_clause] { [statement_list]}

- The method declaration includes:
  - Modifiers: The modifiers are optional. They can be, public, protected, default or private, static, abstract, final, synchronized, throws

Method (contd.)

- Return Type:
  - can be either void or if a value is returned, it can be either a primitive type or a class
  - If the method declares a return type, then before it exits it must have a return statement
- Method Name:
  - The method name must be a valid Java identifier
- Parameter List:
  - Contains zero or more type/identifier pairs make up the parameter list
  - Each parameter in parameter list is separated by a comma
- Curly Braces:
  - The method body is contained in a set of curly braces (opening '{' and closing '}')

Method Example

```
class Circle {
        float pi = 3.14f;
        float radius;
        void setRadius(float rad)       {
                radius = rad;}
                float calculateArea()   {
                float area = pi* radius*radius;
                return (area);
}       }
```

Method invocation

- Methods cannot run on their own, they need to be invoked by the objects they are a part of
- When an object calls a method, it can pass on certain values to the methods (if methods accept them)
- The methods can also return values from themselves if they wish to
- Data that are passed to a method are known as arguments or parameters

Method Invocation

Function Definition

- **Formal Parameters:** The identifier used in a method to stand for the value that is passed into the method by a caller
- **Actual Parameters:** The actual value that is passed into the method by a caller

Function Call

- The number and type of the actual and formal parameters should be same for a method
- In Java, all values are **passed by value**. This is unlike some other programming languages that allow pointers to memory addresses to be passed into methods

Method Invocation

```
class CallMethod {
public static void main (String args[]) {
   float area1;
   Circle circleobj = new Circle();
   circleobj.setRadius(3.0f);
   area1 = circleobj.calculateArea();
   System.out.println("Area of Circle = " + area1);
 }}
```

Constructors

- Java has a mechanism, known as constructor, for automatically initializing the values for an object, as soon as the object is created
- Constructors have the same name as the class it resides in and is syntactically similar to a method
- It is automatically called immediately after the object for the class is created by **new** operator
- Constructors have no return type, not even void, as the implicit return type of a class' constructor is the class type itself
- Types of Constructors: Implicit/Default, Explicit, Parameterized

Constructors

```
Circle() {
}

Circle(double newRadius) {
  radius = newRadius;
}
```

Constructors are a special kind of methods that are invoked to construct objects.

Constructors, cont.

A constructor with no parameters is referred to as a *no-arg constructor*

- Constructors must have the same name as the class itself
- Constructors do not have a return type—not even void
- Constructors are invoked using the new operator when an object is created
- Constructors play the role of initializing objects

Creating Objects Using Constructors

```
new ClassName();
```

Example:

```
new Circle();
```

```
new Circle(5.0);
```

Default Constructor

- A class may be defined without constructors
- A no-arg constructor with empty body is implicitly defined in the class
- This constructor, called *a default constructor*
  - is provided automatically *only if no constructors are explicitly defined in the class*

Static keyword

- Different objects, variables and methods will occupy different areas of memory when created/called
- Sometimes we would like to have multiple objects, share variables or methods
- The *static* keyword effectively does this for us
- Static keyword can be applied to variables/methods and blocks of code.
- Java supports three types of variables: Local, Instance and Class variables
  - Local variables are declared inside a method, constructor, or a block of code
  - Instance variable are declared inside a class, but outside a method

Static keyword

- Class/static variables declaration is preceded with a *static* keyword. They are also declared inside a class, but outside a method
- The most important point about static variables is that there exists only one a single copy of static variables per class
  - The effect of doing this is that when we create multiple objects of that class, every object shares the *static* variable i.e. there is only one copy of the variable declared as *static*. We can declare a variables as static as under:

```
static int var = 0;
```

Instance Variables vs. Class Variables

- All instances of the class share the *static* variables of the class
- A class variable can be accessed directly with the class name, without the need to create an instance
- Without the *static* keyword, it's called "*instance* variable", and each instance of the class has its own copy of the variable

Static Methods

- Like static variables, we do not need to create an object to call our static method
- Simply using the class name will suffice
- Static methods however can only access static variables directly
- Variables that have not been declared static cannot be accessed by the static method directly
- i.e. the reason why we create an object of the class within the main (which is static) method to access the instance variables and call instance methods
- To make a method static, we simply precede the method declaration with the static keyword

Static Methods

```
static void aMethod(int param1) {
        .....
        .....
        }
```

- When we declare a method static, we are basically saying that there should only be one instance of this method within our program (e.g, as in the main method)
- Methods can also be declared with the static keyword
  - static int computeArea(int length, int width) { }

Static initialization block

- A block of statements with static keyword applied to it.
- used for initializing static or class variables.
- in case some logic is used for assigning values to the variables, static blocks can be used.
- The syntax for static block is as follows:

```
static{
...
}
```

Static initialization block

- The static executes as soon as the class loads even before the JVM executes the main method
- There can be any number of static blocks within the class and they will be executed in the order in which they have appeared in the source code

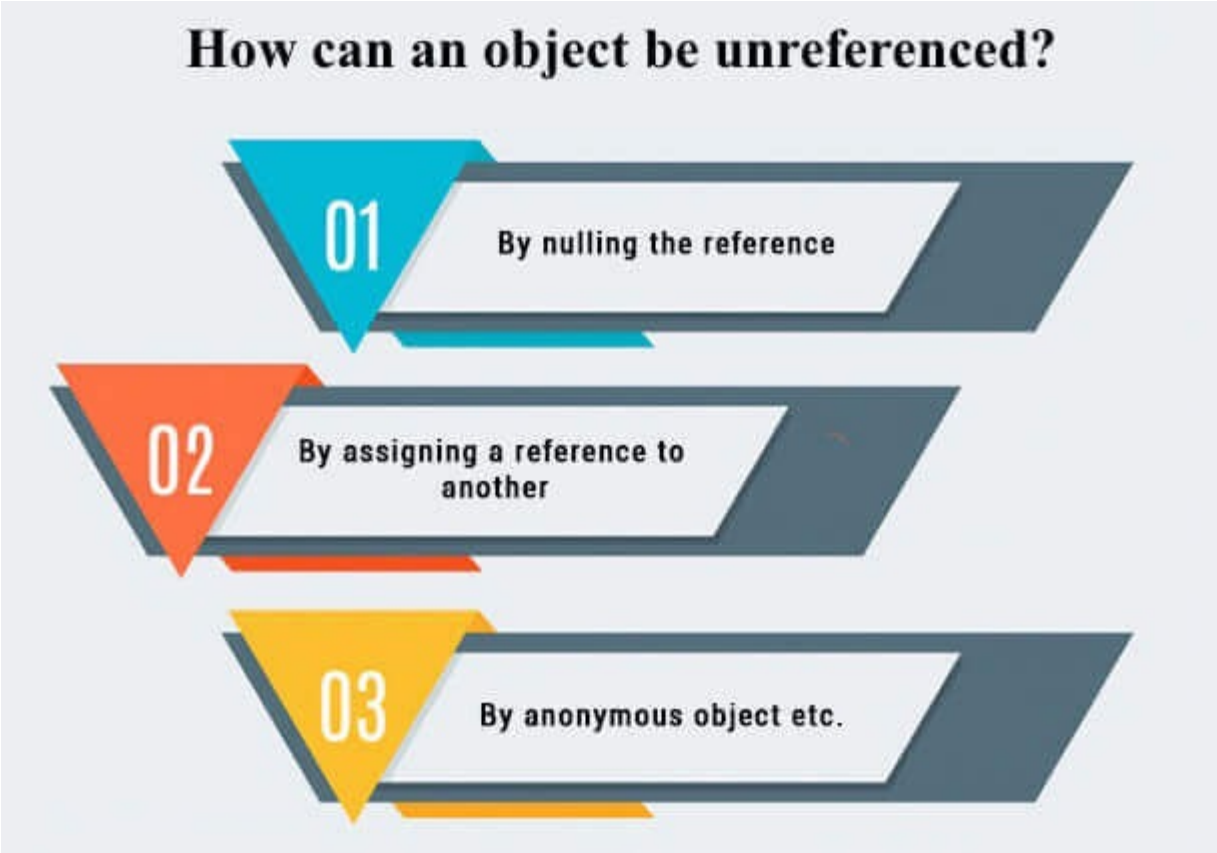
Instance initialization block

- In case the static keyword is dropped from this block, it becomes an instance initialization block
- Actually the code of instance initialization block is placed in the <init> method, which is created for every constructor by the compiler, before the source code mentioned by programmer in the constructor

Automatic Garbage Collection

- The Garbage Collector (GC) collects and removes unreferenced objects from the heap area
- It is the process of reclaiming the runtime unused memory automatically by destroying them
- Garbage collection makes Java memory efficient because it removes the unreferenced objects from heap memory and makes free space for new objects
- It involves two phases:
  - **Mark -** in this step, the GC identifies the unused objects in memory
  - **Sweep -** in this step, the GC removes the objects identified during the previous phase

Unreferenced Objects



Unreferenced Objects

- By nulling a reference

```
Employee e=new Employee();
e=null;
```

- By assigning a reference to another

```
Employee e1=new Employee();
Employee e2=new Employee();

e1=e2;//now the first object referred by e1 is available for garbage collection
```

- By anonymous object

```
new Employee();
```