Chapter 5: Classes and Objects in Depth

Getters, Setters and Constructors
How Private Attributes could be Accessed

- Private attributes are not accessible from outside.
  - Except from objects of the same class.

- They are accessible:
  - From inside: from the object containing the data itself.
  - From objects of the same class.

- They are accessible from outside using accessor operations:
  - Getters
  - Setters
```java
class Course {
    // Data Member
    private String studentName;
    private String courseCode;
}

class CourseRegistration {
    public static void main(String[] args) {
        Course course1, course2;
        // Create and assign values to course1
        course1 = new Course();
        course1.courseCode = "CSC112";
        course1.studentName = "Majed AlKebir";
        // Create and assign values to course2
        course2 = new Course();
        course2.courseCode = "CSC107";
        course2.studentName = "Fahd AlAmri";
        System.out.println(course1.studentName + " has the course " + course1.courseCode);
        System.out.println(course2.studentName + " has the course " + course2.courseCode);
    }
}
```
Getters

The object point of view
• Are operations performed by the object returning outsiders data retrieved from the object state.

The user point of view
• Are services called from outside allowing to retrieve data from the object state.

Getters are:
• Public
• With no parameters
• With return value
Template for Getters

```java
public class ClassName {

    private dataType1 attribute1;
    ...
    private dataType2 attributen;
    ...

    public dataType1 getAttribute1() {
        return attribute1;
    }
    ...
    
    public dataType2 getAttributen() {
        return attributen;
    }
    ...
}
```
Setters

The object point of view
- Are operations performed by the object allowing to receive and store in the object state the data provided by outsiders.

The user point of view
- Are services used by outsiders allowing to provide to the object the data that should be stored in the object state.

Setters are:
- Public
- With 1 parameter
- With no return value
Template for Setters

```java
public class ClassName {
    private dataType1 attribute1;
    private dataType2 attribute2;
    private dataTypeN attributen;

    public void setAttribute1(dataType1 param) {
        attribute1 = param;
    }

    public void setAttributen(dataTypeN param) {
        attributen = param;
    }
}
```
public class Course {

    // Attributes
    private String studentName;
    private String courseCode;

    ... 

    public String getStudentName() {
        return studentName;
    }

    public String getCourseCode() {
        return courseCode;
    }

    ...

    public void setStudentName(String val) {
        studentName = val;
    }

    public void setCourseCode(String val) {
        courseCode = val;
    }
}

```java
public class CourseRegistration {
    public static void main(String[] args) {
        Course course1, course2;
        //Create and assign values to course1
        course1 = new Course();
        course1.setCourseCode("CSC112");
        course1.setStudentName("Majed AlKebir");
        //Create and assign values to course2
        course2 = new Course();
        course2.setCourseCode("CSC107");
        course2.setStudentName("Fahd AlAmri");
        System.out.println(course1.getStudentName() + " has the course " + course1.getCourseCode());
        System.out.println(course2.getStudentName() + " has the course " + course2.getCourseCode());
    }
}
```
Passing an Object to a Setter

```java
LibraryCard card2; // Passing side

// Passing an instance of Student
card2 = new LibraryCard( );

// Setting the owner of the card
card2.setOwner(student);

class LibraryCard {
    public void setOwner(Student student) {
        owner = student;
    }
}
```

Memory Allocation:

1. Passing side: `student` is passed to `card2`.
2. Receiving side: `card2` is set with `student` as its owner.
Setters and Sharing Objects

- The same Student object reference is passed to card1 and card2 using setters.

```java
Student student;
LibraryCard card1, card2;

student = new Student();
student.setName("Jon Java");
student.setEmail("jj@javauniv.edu");

card1 = new LibraryCard();
card1.setOwner(student);
card1.checkOut(3);

card2 = new LibraryCard();
card2.setOwner(student); // the same student is the owner of the second card, too
```

- Since we are actually passing the same object reference, it results in the owner of two LibraryCard objects referring to the same Student object.
Class Constructors

• A class is a blueprint or prototype from which objects of the same type are created.

• Constructors define the initial states of objects at birth.
  • `ClassName x = new ClassName();`

• A class contains at least one constructor.

• A class may contain more than one constructor.
The Default Class Constructor

- If no constructors are defined in the class, the default constructor is added by the compiler at compile time.

- The default constructor does not accept parameters and creates objects with empty states.
  - `ClassName x = new ClassName();`
Class Constructors Declaration

- The **constructor name**: a constructor has the same names as the class.

- The **parameters** represent values that will be passed to the constructor for initialize the object state.

- Constructor declarations look like method declarations except that:
  - they use the name of the class
  - and have no return type.
**Example of a Constructor with No-Parameter**

```java
public class Kasree {
    private int bast;
    private int maquam;
    public Kasree() {
        bast = 0; maquam = 1;
    }
    ...}
```

A. The instance variable is allocated in memory.

B. The object is created with initial state

C. The reference of the object created in B is assigned to the variable.

**State of Memory**

<table>
<thead>
<tr>
<th>Object: Kasree</th>
</tr>
</thead>
<tbody>
<tr>
<td>bast</td>
</tr>
<tr>
<td>maquam</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object: Kasree</th>
</tr>
</thead>
<tbody>
<tr>
<td>bast</td>
</tr>
<tr>
<td>maquam</td>
</tr>
</tbody>
</table>
Class with Multiple Constructors

public class Kasree {
    private int bast;
    private int maquam;

    public Kasree() {
        bast = 0; maquam = 1;
    }
    public Kasree(int a, int b) {
        bast = a;
        if (b != 0) maquam = b;
        else maquam = 1;
    }
    ...
}

Code

Kasree x, y;
x = new Kasree();
y = new Kasree(4, 3);

State of Memory

A. The constructor declared with no-parameter is used to create the object

Object: Kasree

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bast</td>
<td>0</td>
</tr>
<tr>
<td>maquam</td>
<td>1</td>
</tr>
</tbody>
</table>

B. The constructor declared with parameters is used to create the object

Object: Kasree

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>bast</td>
<td>4</td>
</tr>
<tr>
<td>maquam</td>
<td>3</td>
</tr>
</tbody>
</table>
Overloading

- Two of the components of a method declaration comprise the *method signature*:
  - the method's name
  - and the parameter types.

- The signature of the constructors declared above are:
  - Kasree()
  - Kasree(int, int)

- *Overloading* methods allows implementing different versions of the same method with different *method signatures*:
  - This means that methods within a class can have the same name if they have different parameter lists.
Overloading (cont.)

- Overloaded methods are differentiated by:
  - the number,
  - and the type of the arguments passed into the method.

- You cannot declare more than one method with:
  - the same name,
  - and the same number and type of parameters.

- The compiler does not consider return type when differentiating methods.
  - No declaration of two methods having the same signature even if they have a different return type.
Intra-Constructors Calls

- A constructor of a class may use an other constructor of the same class.

```java
public class Kasree {
    private int bast;
    private int maquam;

    public Kasree(int a, int b) {
        bast = a;
        if (b != 0) maquam = b;
        else maquam = 1;
    }

    public Kasree() {
        Kasree(0, 1);
    }

    x = new Kasree();
    y = new Kasree(4, 3);
}
```