CHAPTER 3

BASICS OF POO

Contents

Key Objectives

The key objectives of this chapter are to introduce the fundamental concepts of object-oriented programming (OOP) and to help student understand how to create and use classes and objects *in Java. By the end of the chapter, student should be able to define classes with attributes and* **methods create objects from these classes and understand the relationship between classes and their instances. To reinforce these concepts and give student hands-on experience the chapter concludes with a series of practical activities allowing him to apply what he has learned and** *build confidence in working with classes and objects.*

3.1 Classes and Objects

Recapitulation 3.1

- **Classes and objects are the two main aspects of object-oriented programming. A class is a template for objects and an object is an instance of a class.**
- **To create a class we use the keyword `class` as illustrated by the follwing example where *we define a class named Student. Inside the class, we declare variables (attributes) and* methods.**

```
1  public class Student
2      int age
3      String name
4      void display() System.out.println(name+ +age)
5
```

Listing 3.1: Student class in Java

- **An object is created from a class. We have already created the class named Student so now we can use this to create objects. To create an object of Student specify the class name followed by the object name and use the keyword `new`:**

```
1  public static void main(String args)
2    Student myObj new Student()
3    System.out.println(myObj.name)
4    myObj.display()
5
```

Listing 3.2: Creation of an object of the Student class in Java

3.2 Attributes and Methods

Recapitulation 3.2

- **In the class Student (Listing 3.1) we declared two variables age and name. They are actually attributes of the class (class attributes are variables within a class). They** *represents the state of an object and reflects the properties of an object.*

- *lass methods are functions defined within a class that describe the behaviors or actions* **that objects of the class can perform. These methods can manipulate object data perform operations and return results. As example in Listing 3.1 the display() method is an instance method that displays the information of a student (age and name).**
- **We can access attributes and methods by creating an object of the class and by using the dot syntax as used in the previous example (Listing 3.8).**

Recapitulation 3.3

- **In a class Java we can have multiple methods with the same name this is called method overloading.**
- *he key distinction between overloaded methods lies in their parameter lists. This difference can be achieved in different ways:*
  - ***Different Number of Parameters*: Methods can have the same name but accept a varying number of arguments.**
  - ***Different Data Types of Parameters*: Methods can have the same name and the** *same number of parameters, but the data types of those parameters must differ.*
  - ***Different Order of Parameters*** *(with different types): If the data types are different,* **changing the order of parameters can also lead to method overloading.**

Recapitulation 3.4

Java always uses "pass by value" for method parameters, but this has different consequences **depending on whether the parameter type is simple (int double etc.) or advanced (object).**

① **When a primitive type variable is passed to a method a copy of its value is made and** *assigned to the method's formal parameter. Any modifications to this parameter within the method will only affect the local copy, and the original variable in the calling code* **remains unchanged. Let s consider the Listing 3.3 values of variables x and y remained the same after calling modify methods.**

② **When an object is passed to a method (such as array Arraylist Person Student ...) a copy of the object s reference (memory address) is passed by value. This means both the original variable and the method parameter refer to the same object in memory. If** *any modification of the state of the object (change a field's value) through the passed reference, these changes will be reflected in the original object because both references*

point to the same object. Let s consider the Listing 3.3, *value of the first element of* the array tab changed after calling modify method. However changes on tab itself (reassign the parameter to a new object within the method a new int 5)) *does not affect* the original variable outside the method. This is because changes made within the method to the reference itself are not visible outside the method.

```
1  public class PassingArguments
2    static void modify(int a) a 12
3    static void modify(int a int b)
4      int z   a
5      a  b
6      b   z
7    static void modify(int     a) a 1    0  a    new int 5
8    public static void main(String   args)
9
10     int x 10   int y 8
11     int tab    3 4 5
12     System out.print( Before modification:   )
13     System out.println(x+   +y+   +tab 1 +   +tab.length)
14     modify(x)  modify(x y)
15     modify(tab)
16     System out.print( After modification :   )
17     System out.println(x+   +y+   +tab 1 +   +tab.length)
18     // Execution output:
19     // Before modification: 10 8 4 3
20     // After modification : 10 8 0 3
21
```

Listing 3.3: Example of passing parameters in Java

3.3 Constructors

Recapitulation 3.5

- **Constructor is a special method used to initialize new objects of a class. It has the same name as the class. The constructor is called when an object of a class is created. Constructors can also take parameters which is used to initialize attributes.**
- *onstructors can also take parameters, which is used to initialize attributes with specific* **values provided during object creation.**
- *class can have multiple constructors, as long as they have different parameter lists (different number of parameters, different types of parameters, or different order of parameter* **types). This allows for various ways to initialize an object.**
- **Unlike methods constructors do not have a return type not even void.**
- *f a class does not explicitly define any constructors, Java automatically provides a default,* **no-argument constructor. This default constructor initializes instance variables with their default values (e.g. 0 for numeric types null for object references false for booleans).**
- *f any custom constructor is defined (with or without arguments) in a class, Java will not* **automatically provide the default constructor. If we need a no-argument constructor in** *such a case, we must explicitly define it.*
- **We can add contructors to the Student class as follows:**

```
1  public class Student
2    int age
3    String name
4    Student(int a  String n) age a  name n   // Custom constructor
5    Student() age 20  name  Ahmed     // a no-argument constructor
6    void display() System out println(name+   +age)
7
```

Listing 3.4: Student class in Java with two constructors

3.4 Static Attributes and Methods

Recapitulation 3.6

- **The static keyword means that a member of a class (attribute or method) belongs to the *class itself, rather than to any specific instance of that class. Thus, we can access static* members without creating an instance of an object.**
- **When we declare an attribute static exactly a single copy of that attribute is created and shared among all instances of that class (The value of this static attribute is shared across all objects of the same class).**
- **A static attribute can be used for example to track the number of objects created. This allows the counter to be incremented with each new object. As we can see in the Listing 3.5 the static variable numberOfStudent will be incremented each time we instantiate the Student class.**

```
1       public class Student
2         int age
3         String name
4         static int numberOfStudent  // static variable
5         Student(int a  String n) age a  name n  numberOfStudent++
6         Student() age 20  name  Ahmed  numberOfStudent++
7         void display() System.out.println(name+  +age)
8
```

Listing 3.5: Student class in Java with a static attribute

Let s create two Student objects and expect the counter to have a value of two:

```
1       public static void main(String  args)
2         Student s1  new Student()
3         Student s2  new Student()
4         System.out.println(Student.numberOfStudent)  // 2
5         myObj.display()
```

Listing 3.6: Creation of two objects of Student class in Java

- **Static variables can be accessed through an instance (s1.numberOfStudent) or directly from the class (Student.numberOfStudent).**

- Similar to static attribute static methods also belong to a class instead of an object. So we can invoke them without instantiating the class. enerally static methods are used to perform an operation that s not dependent upon instance creation.
- Static methods can be used to create utility or helper classes. A popular example is Math utility class (Mat h. s rt( ) Mat h. pow( )).

R

Instance methods can directly access both instance methods and instance variables
Instance methods can also access static variables and static methods directly
Static methods can access all static variables and other static methods
Static methods can t access instance variables and instance methods directly. They need some object reference to do so.

3.5 Visibility and Accessor Methods

Recapitulation 3.7

- The visibility of methods and properties within a class determines where they can be accessed. *This is controlled by access modifiers such as* `public` (accessible from anywhere) `private` *(only accessible within the class that defines them),* `protected` (accessible within *the defining class and by any class that inherits from it), and default (accessible within* the same package).
- To retrieve and update the value of a private attribute of a class getter and setter methods are used.
- The get method returns the variable value and the set method sets the value. Syntax for both is that they start with either get or set followed by the name of the variable with the *first letter in upper case. Consider Student class again, but this time with private instance* variables for name and age:

```
1     public class Student
2       privare int age
3       private String name
4       static int numberOfStudent  // static variable
5       Student(int a String n) age a  name n  numberOfStudent++
```

```
6         Student() age 20  name  Ahmed   numberOfStudent++
7
8         //  etter method for name
9         public String getName()  return name
10         // Setter method for name
11        public void setName(String name)  this.name   name
12        //  etter method for age
13        public int getAge()  return age
14        // Setter method for age with check of validation
15        public void setAge(int age)
16          if (age   0)  // Ensure age is a positive value
17            this.age   age
18            else
19            System.out.println( Invalid age  age must be a positive number. )
20
21
22
23        void display() System.out.println(name+   +age)
24
```

Listing 3.7: Student class with private attributes

In this case because Name attribute is private within the class Student the access to this attribute outside of class Student is doing by using get method for retrieving its value and using set mathod for updating its value. As example Listing 3.8 becomes as follows:

```
1  public static void main(String   args)
2     Student  myObj   new Student()
3     // Because Name attribute is private within the class  the access to it is  by
      using get and set methods
4     myObj.setName( Mohammed )
5     System.out.println(myObj.getName())
6     myObj.display()
7
```

Listing 3.8: Retrieve and update value of age field outside of class Student

3.6 Quick Check

Quick Check 3.1 Which of the following constructors of a class named ClassA are invalid?

①
```
public int ClassA(int one)
...
```

②
```
public ClassA(int one, int two)
...
```

③
```
void ClassA()
...
```

Quick Check 3.2 Which statement about instance methods is correct?

① They can only be accessed from static methods.

② They require an object to be called.

③ They are automatically synchronized

④ *They must be declared final*

Quick Check 3.3 What will happen when calling a static method using an object?

① Runtime error

② Compilation error

③ Runs without errors

④ NullPointerException

Quick Check 3.4 What is the correct way to create an object called myObj of MyClass?

① `class myObj = new MyClass();`

② `class myObj = new myObj();`

③ `new myObj MyClass`

④ `MyClass myObj new MyClass()`

Quick Check 3.5 **What is method overloading?**

① **Two methods with the same name and parameters**

② *Two methods with the same name but different parameters*

③ *Two methods with the same name but different return types*

④ *Defining methods inside a loop*

Quick Check 3.6 **Which of the following is true about Java constructors?**

① **Constructors have a return type of void.**

② **Constructors must always be public.**

③ **A constructor is automatically called when an object is created.**

④ *Constructors can only be defined explicitly.*

Quick Check 3.7 *What happens if you do not define a constructor in a Java class?*

① **The program will not compile.**

② **The compiler provides a default constructor.**

③ **The class cannot be instantiated.**

④ **The constructor from another class will be used.**

Quick Check 3.8 **Can a Java constructor be private?**

① **No it must be public.**

② **es but the class cannot be instantiated outside the class.**

③ **es and it can be accessed from anywhere.**

④ **No Java does not allow private constructors.**

Quick Check 3.9 How can one constructor call another constructor within the same class?

① Using super()

② Using this()

③ Using new()

④ Constructors cannot call each other

Quick Check 3.10 What will be the output of this program?

```
class Ab
  int num
  Ab()
    this(100)
    System.out.println( Default Constructor )

  Ab(int n)
    num  n
    System.out.println( Parameterized Constructor )


public class Main
  public static void main(String  args)
    Ab obj  new Ab()


```

Quick Check 3.11 Consider the following class declaration:

```
class QuestionOne
    public final int A  345
    public int b
    private float c
    private void methodOne( int a)  b  a
    public float methodTwo( )  return 23

```

Identify invalid statements in the following main class. For each invalid statement state why it is invalid.

```
class Q1Main
  public static void main(String   args)
    QuestionOne  1
     1   new QuestionOne( )
     1. A   12
     1. b   12
     1. c   12
     1. methodOne(12)
     1. methodOne( )
    System out. println( 1. methodTwo(12))
     1. c    1. methodTwo( )
```

Quick Check 3.12 Consider the following class:

```
class QuestionTwo
  private int count
  public void init( )   count   1
  public void increment( )   count   count + 1
  public int getCount( )   return count
```

What will be the output from the following code:

```
class Q2Main
  public static void main(String   args)
    QuestionTwo  2        2   new QuestionTwo( )
     2. init()
     2. increment()
     2. increment()
    System out. println( 2. getCount())
```

Quick Check 3.13 Consider the following class:

```
class QuestionThree
 public int count
 public void init( )   count   1
 public int increment( )
   count   count + 1
   return count
```

What will be the output from the following code:

```
class Q3Main
 public static void main(String   args)
   QuestionThree  3
    3   new QuestionThree( )
    3.init()
    3.count    3.increment() +  3.increment()
   System.out.println( 3.increment())
```

3.7 Activities

Exercise 3.1 Creation of Student class

A student will be modeled here by the Student class of a package named Studentmanager x as follow:

The Student class has three attributes:

- Name: String
- Fname: String
- Marks: an arraylist of integers

The Student class has the following methods:

- SetName: to enter the name of the student
- SetFname: to enter the fammily name of the student
- AddMark: to enter a mark of the student
- RemoveMark: to delete the mark i of the student
- Average: compute and returns as value the average of marks of the student
- Display: to display information of the student (name fname and average of marks)
- DisplayMarks: to display marks of the student
- SetMarks: to update the mark i of the students if exist.

The Student class has the following constructors:

- a constructor that allows you to initialize the student s name and Fname (from arguments).
- a constructor that allows you to initialize the student s name and Fname and a mark (from arguments).
- a default constructor that allows you to initialize the student s name as unknown Name and Fname as unknown FName and a mark value 0.

The student class has the following static functions:

- a function named Randm allows to generate a random integer.
- a boolean function named Compare allows to compare two students according to their *average. The function returns true if the average of the first student is greater than the* average of the second one.

In a main class:

① *use the constructors defined above to create two new students* S1 and S2 named Ahmed Ismail and Khadidja Mouhammed . Then give to each one three marks. Finally display these students with their averages and marks.

② create a list StudentList of students. Then add four students to this list. ive to each one three random marks. Finally display these students with their averages and marks.

③ Display the student with the max average the passed and failed students.

Solution

```
package studentmanager
import java.util.ArrayList
import java.util.Scanner
public class Students
   String Name
   String Fname
   ArrayList Integer marks new ArrayList Integer ()

   void setName()
       Scanner in new Scanner(System in) Name in.nextLine()
   void setFname() Scanner in new Scanner(System in) Fname in.nextLine()

   void setName(String s) Name s
   void setFname(String s) Fname s
   void addmark(int m) marks.add(m)
   void removemark(int i) if (i marks.size()) marks.remove(i)
   double average()
       int s 0
       for(int i 0 i marks.size() i++) s s+marks.get(i)
       return s/marks.size()

   void Display() System out.println(Name+ + Fname+ : +average())
   void DisplayMarks()
       for(int i 0 i marks.size() i++) System out.println(i+ : + marks.get(i))

```

```
package studentmanager
package studentmanager
import java.util.Random import java.util.ArrayList
public class StudentManager 3
    static int rand() Random rand new Random() return rand.nextInt(20)
    public static void main(String args)
        Students A new Students()
        A.Name Mohamed
        A.Fname Ibrahim
        for(int i 0 i 3 i++) A.addmark(rand())
```

```
11
12          Students B  new Students()
13          B.setName( Anina )
14          B.setFname( Ali )
15          for(int i 0 i 3 i++) B.addmark(rand())
16
17          A.Display()      A.DisplayMarks()
18          B.Display()      B.DisplayMarks()
19          ArrayList Students StudentList  new ArrayList Students ()
20          StudentList.add(B)    StudentList.get(0).Display()
21
```

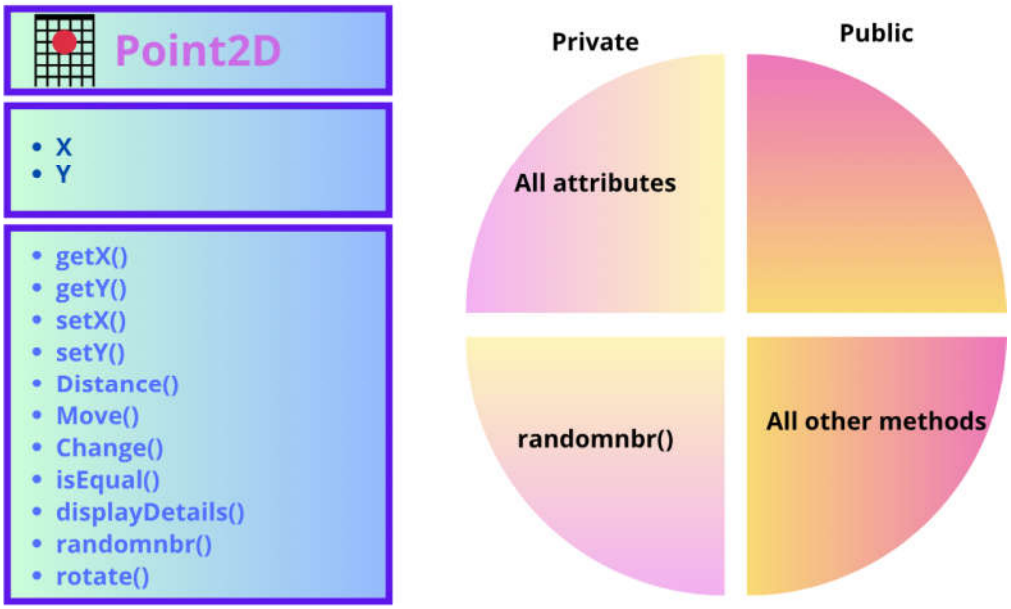

Figure 3.1: Point2D class.

Exercise 3.2 Point2D class

Define a class named Point2D to model a two dimensional point characterized by its coordinates **(x y). The point2D class contains the following constructors and methods (see Figure 3.1):**

① **the getters and the setters for the class attributes.**

② **a default constructor that allows to initialize the two attributes randomly and a constructor** *that allows to initialize a point from two float parameters.*

③ **a method that returns the distance between the current point and the origin of the coordinate**

system.

④ a method that returns the distance between the current point and another point passed as parameters.

⑤ a method named move to move the location of the point by (d_x d_y) on the plane.

⑥ a method named change to change the coordinates of the current point.

⑦ a display method which displays in a well-organized way the properties and distance between the point and the origin point.

⑧ a method named rotate to rotate the point by 90 degrees clockwise around the origin. When point is getting rotated 90 clockwise around the origin the following changes happen to its coordinates: new value of x $\leftarrow$ y new value of y $\leftarrow -$x.

⑨ a static function named isEqual takes as arguments two points and returns true if the two points have the same distance from the origin point otherwise false.

In a main class:

① Create a table T of Point2D and initialize randomly all points of T and display them.

② Create a point P and display the distances between P and all elements of T.

③ Rotate P and display the distances between P and all elements of T.

R All attributes and the random function must be private.

Solution

```
1  package tp_poo
2  import java.util.Random
3  public class Point2d
4    private int x
5    private int y
6
7      public int get()   return x
8      public void set(int x)  this.x  x
9      public void set(int y)  this.y  y
10     public int get()  return y
```

```
11     private int rand()  Random r new Random()  return r.nextInt(10)
12     public Point2d()
13         set (rand())
14         set (rand())
15     //System out.print( A new  point is created : x  +x+  y  +y+  )
16
17   public Point2d(int a int b)
18       x a    y b
19     //System out.print( A new  point is created : +x+  +y+  )
20
21 double distance()  return Math.s rt(x x+y y)
22 double distance(Point2d p) return Math.s rt((x-p.x) (x-p.x)+(y-p.y) (y-p.y))
23
24   void print() System out.println(  x  +x+  y  +y+  )
25   void printAll() System out.print(  x  +x+  y  +y+  distance  + distance())
26
27     void change(int a int b) x a y b
28     void move(int dx int dy) x x+dx y y+dy
29     void rotate()  int t x  x y y -x
30
31   static boolean isE uel(Point2d a Point2d b) return a.distance()  b.distance()
32   Point2d Merge(Point2d p)  return new Point2d(x+p.x Math.max(y p.y))
33
```

Exercise 3.3 Class Circle

Using the class of Exercise 3.2 (Point2D) and in the same project (application) create a new class named Circle characterized by its center point (which is a Point2D) and a radius.

R All attributes must be private.

The Circle class contains the following constructors and methods:

① the getters and the setters for the class attributes.

② a default constructor that allows to initialize attributes randomly and a constructor that

allows to initialize a circle from two parameters (center and radius) a constructor that allows to initialize a circle from one parameter (radius) and the center as the origin point.

③ Area method that returns the area of the circle (Area $\Pi * R^2$)

④ perimeter method that returns the perimeter of the circle (perimeter $2 * \Pi * R$)

⑤ isInside method that returns true if a point is inside the circle false otherwise.

⑥ isEqual method that compares the current circle with another circle passed as parameter. The isEqual method returns true if the two circles have the same area false otherwise.

⑦ display method which displays in a well-organized way the properties the area and the perimeter of the Circle.

⑧ merge method that takes as parameter a circle C and returns as value a new circle which is the result of merging the current circle with C such that the new circle s center is the point center that has the max distance from the origin and the radius is the max one also.

In a main class:

① Create two circles and display them

② Create a table Tp of Point2D and initialize it randomly then diplay all points that are inside the previous circles.

Exercise 3.4 Complex numbers

We want to implement a class that represents complex numbers and supports basic operations. Complex numbers are numbers that can be written in the following form z a + bi where a represents the real part i b represents the imaginary part a and b are real numbers and i is an imaginary unit called iota that represents $\sqrt{-1}$ and $i^2 \quad -1$. Create a Java class named Complex that models complex numbers. Complex class must include necessary attributes constructors getters/setters and provides display method to print a complex number in clear format as well as isEqual method that allows to compare two complex numbers. In addition Complex class must supports the following operations:

- Addition: (a + bi) + (c + di) (a + c) + (b+ d)i (Addition method returns the sum of the current complex number and another complex number).

- Subtraction: (a + bi) − (c + di) (a − c) + (b − d)i (Subtraction method returns the subtraction of the current complex number and another complex number).
- Multiplication: (a + bi)(c + di) (ac − bd) + (ad + bc)i (Multiplication method returns the multiplication of the current complex number and another complex number).

Exercise 3.5 Pet class

Write a Java program to create a class called Pet with attributes and methods as illustrated by Figure 3.2:

- method grow increases the age s pet by 1.
- method speak prints a generic pet sound .
- *ethod humanAge returns the age of the pet and displays "Pets can age differently depending* on their species breed and size .
- method size returns as value the pet size (small medium large or iant) depending on the weight of the pet.

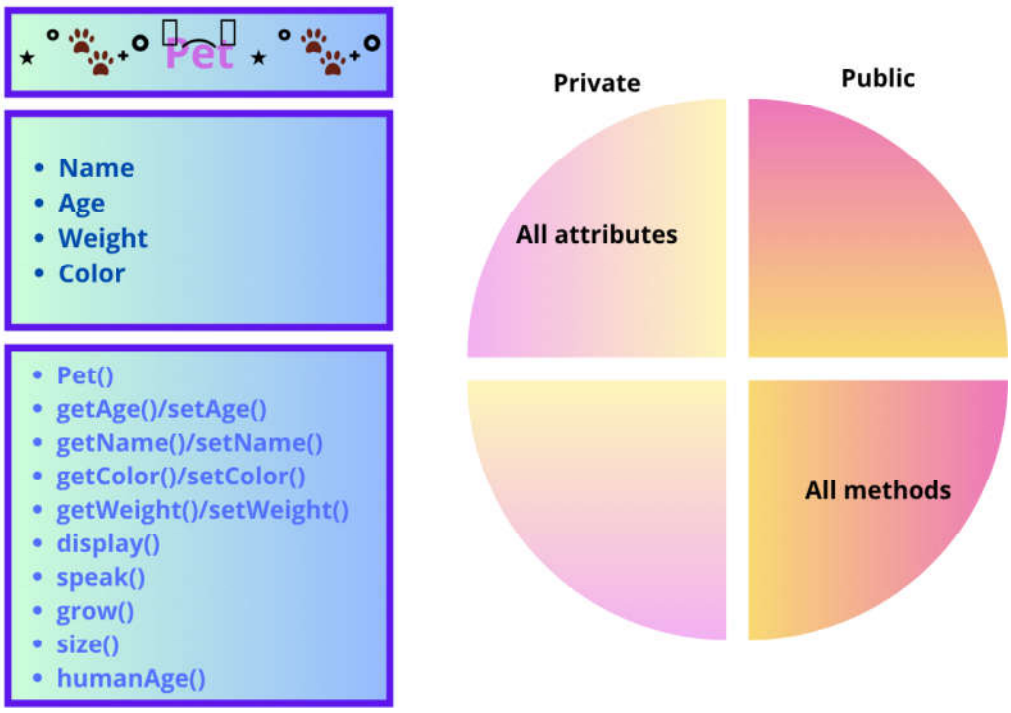

Figure 3.2: Pet class

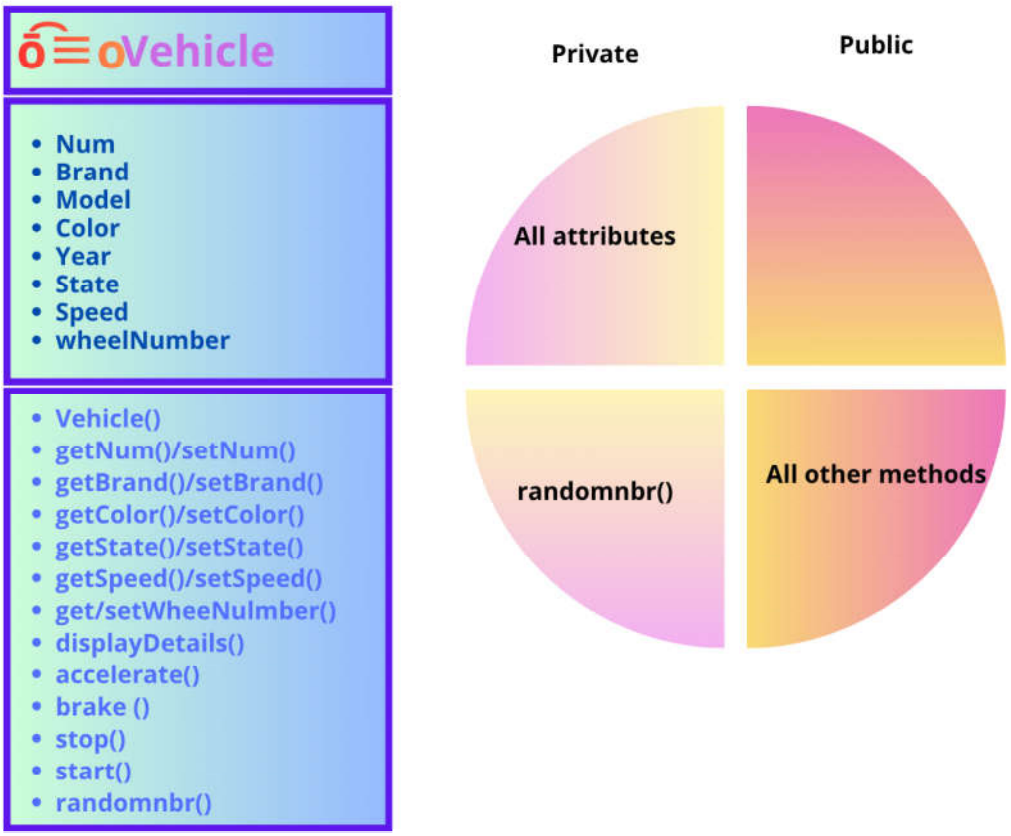

Figure 3.3: Vehicle class

Exercise 3.6 Class vehicle

Write a Java program to create a class called Vehicle with attributes and methods as illustrated by Figure 3.3:

- method accelerate increases the vehicle s speed and takes the increment value as parameter as it can use a random one.
- method brake decreases the vehicle s speed and takes the decrement value as parameter as it can use a random one.

Exercise 3.7 Driver class

Write a Java program to create a class called Driver . The Driver class uses a Vehicle object as part of its data (A Driver has a Vehicle). Class Driver contains:

- Driver s name
- A Vehicle object
- A constructor that initializes both the driver s name and the vehicle.

- A method displayDriverInfo() that displays the driver s name and the vehicle information.

In a test program (Main class):

- Create a Vehicle object.
- Create a Driver object that uses that vehicle
- Display all the information about the driver and their vehicle
- Create multiple Driver and Vehicle objects and store them in an array or list.

Exercise 3.8 Polynome2D class

Create a class Polynome2D that represents a second-degree polynomial: $ax^2 + bx + c$ where $a\ b\ c \in R\ a \neq 0$ by breaking down the code and methods into auxiliary sub-methods in all *possible cases. The class must contain private fields, necessary constructors and necessary getters* and setters. In addition Polynome2D class provides the following methods:

- Display: prints the polynomial in clear format.
- solve() which computes the two roots of the polynomial in R.
- solveComplex() which computes the two roots of the polynomial in C (refer to Exercise 3.4).

Reminder:

- $= b^2 - 4 * a * c$
- x_1 $\frac{-b-\sqrt{\Delta}}{2*a}$
- x_2 $\frac{-b+\sqrt{\Delta}}{2*a}$
- z_1 $\frac{-b-i\sqrt{|\Delta|}}{2*a}$
- z_2 $\frac{-b+i\sqrt{|\Delta|}}{2*a}$.

Exercise 3.9 Temperature class

Suppose you want to maintain the temperatures for each day of a month. To do this:

① Write a Temperature class that allows to store a temperature in Fahrenheit with the necessary getters/setters and constructors that allow to initialize the temperature to 20 or accept a temperature in Fahrenheit (in double) as a parameter.

② The Temperature class allows basic conversions to Celsius (getCelsius) and to Kelvin

(getKelvin) such that:

$$Celsius \quad (5\ 9) * (Fahrenheit - 32)$$

$$Kelvin \quad ((5\ 9) * (Fahrenheit - 32)) + 273$$

③ The Temperature class provides a method to display the temperature in Fahrenheit.

④ The Temperature class provides a method to display the temperature based on a parameter: C′ in Celsius F′ in Fahrenheit and k′ in Kelvin.

⑤ Write a class TemperatureMonth that stores the temperatures of a month as an array of Temperature values. The TemperatureMonth class has constructors that initialize the month and its temperatures. By default the temperatures for the month are set to 20 however they *may be initialized to the value specified by the parameter.The month and its number of days* can be passed as parameters by default is 30 and the month name is June.

⑥ The temperatures are accessed using two methods: `setTemperature(double tmp int day)` and `getTemperature(int day)`. If the day value passed as a parameter is negative or greater than the number of days in the month an error message should be displayed.

⑦ The TempertureMois class has a display method that displays the temperatures of the month day by day in Celsius.

⑧ Write a test class named TestTemperature whose purpose is to validate the correct functioning of the Temperature and TemperatureMois classes:

① Create an instances of the Temperature class named T1 using the default constructor (initialized with the default temperature value). Then display it in kalvin in Celsius and in Fahrenheit.

② Create an instances of the Temperature class named T2 using a parameterized constructor (initialized with the temperature value provided as an argument). Then display it in kalvin in Celsius and in Fahrenheit.

③ Create an instances of the TemperatureMois class m using the default constructor (initialize all monthly temperature values to the default value). Then modify values of TM using double random values. Display these values in Celsius.

Solution

```
public class Temperature
  double t
  public Temperature()    t 20
  public Temperature(double t)  this.t    t
  public double getT()  return t
  public void setT(double t)    this.t    t
  public double getTcelus()    return (0.5555 (t-32))
  public double getTKelven()  return ((0.5555 (t-32))+273)
  public void Print()    System.out.println( Fernheit    +t)
  public void Print(char c)
    switch (c)
      case  c :
      System.out.println( Celius    +getTcelus())  break
      case  k :
      System.out.println( Kelven    +getTKelven())  break
      case  f :
      System.out.println( Fernheit    +getT())  break
      default:System.out.println( Please inter c  k or f    )

```

Listing 3.9: Temperature class

```
public class Tmois
  Temperature    m
  String mois

  public Tmois()
    m  new Temperature 30    mois  June
    for(int i 0 i mlength i++)mi   new Temperature()


  public Tmois(String mmint n double t)
    m  new Temperature n    mois mm
    for(int i 0 i mlength i++)mi   new Temperature(t)


```

```
15      public void setTemperteur(int i  double t)
16        if((i  mlength)   i 0) System out println( error )
17        else this.mi .setT(t)
18
19      public  double getTemperteur(int i)  return mi .getT()
20
21      public void print()
22        for(int i 0 i mlength i++)
23          System out print(i+   )
24          mi .Print( c )
25
26
```

Listing 3.10: Tmois class

```
1  public class TestTemperature
2  public static void main(String   args)
3    Temperature T1 new Temperature()
4      T1. Print( c )
5      T1. Print( k )
6      T1. Print( f )
7    Temperature T2 new Temperature(45)
8      T2. Print( c )
9      T2. Print( k )
10     T2. Print( f )
11
12   Random nb new Random()
13   Tmois m new Tmois()
14   for(int i 0 i m mlength i++)
15     m setTemperteur(i  nb. nextDouble(100))
16   m print()
17
18
```

Listing 3.11: TestTemperature class