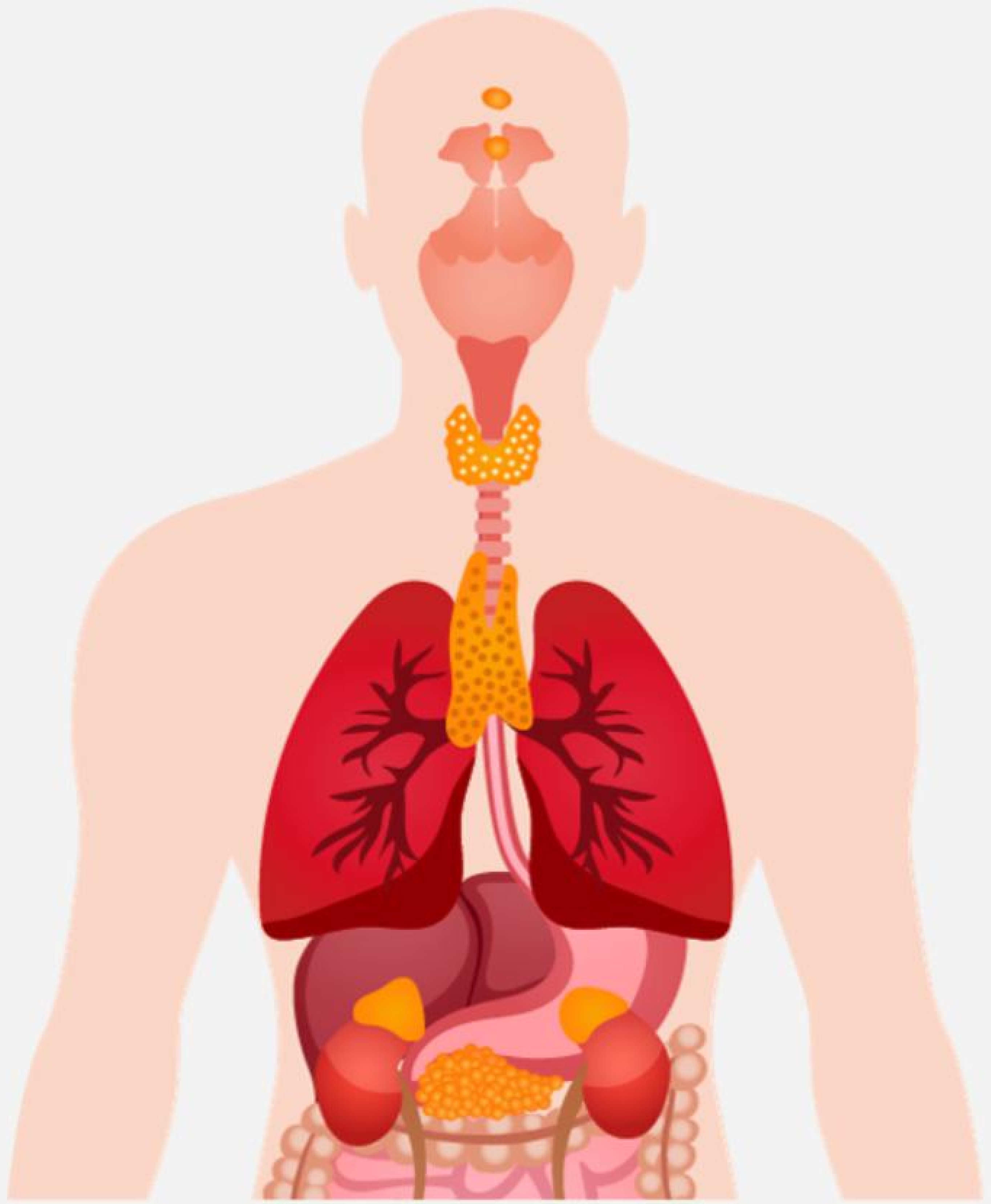
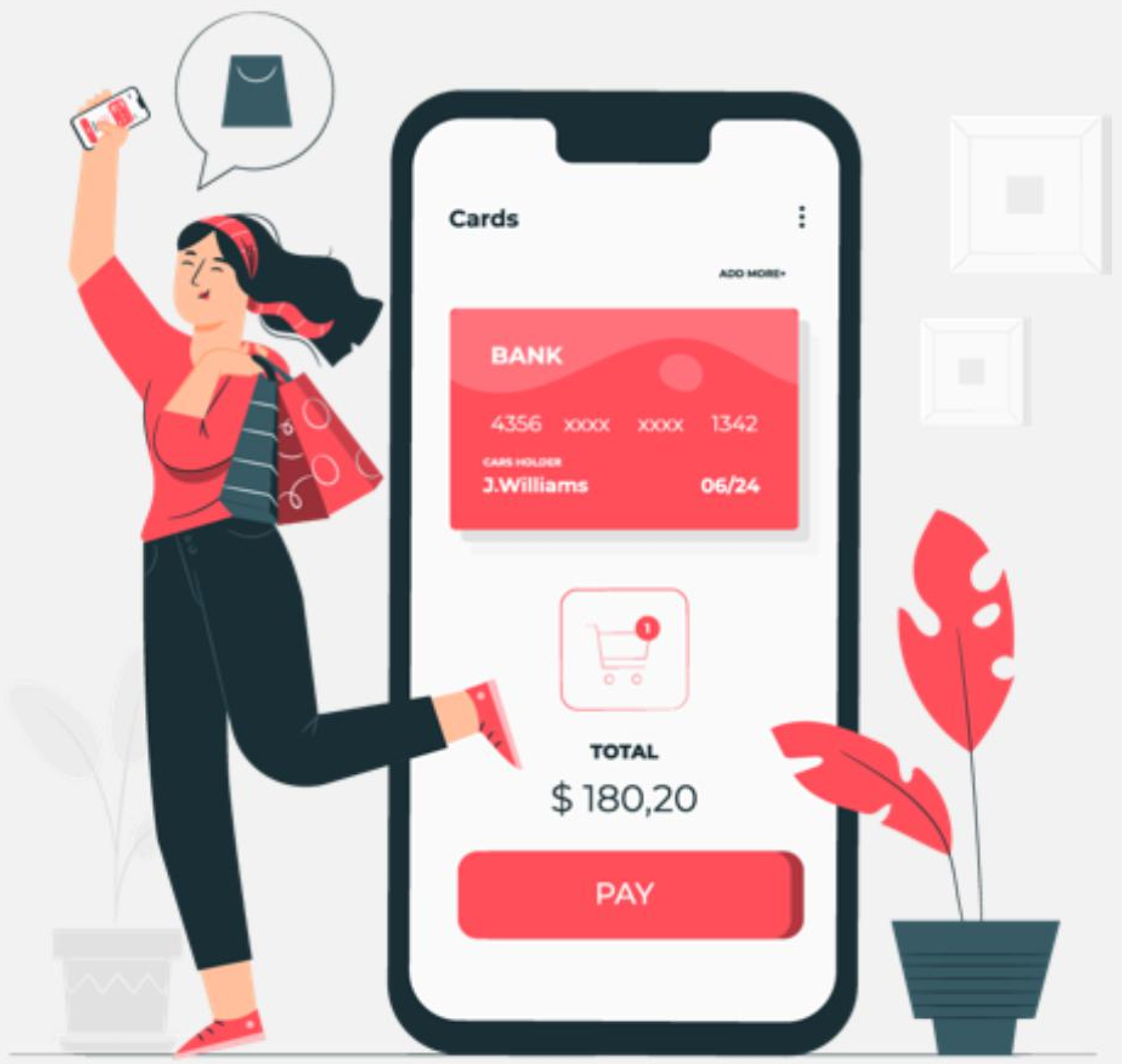
Think and Tell

How does our body work?





Digital Wallet

- When was the last time you used your digital wallet?
- Why do you use a digital wallet?

Digital Wallet (contd.)

These days, digital wallets are commonly used for the purchases we make.

- How does the payment process work on your cell phone?

Digital Wallet (contd.)

- What features of a digital wallet are hidden from the end users?

Smart Speakers

- Have you ever seen a smart speaker work?
- How does it respond to your instructions?

Menu

00:05 00:00:44 05/ 44

Smart Speakers (contd.)

How does the voice control mechanism of a smart speaker work?

Think and Tell

- A startup company wants to manage its HR operations. As a software programmer, you need to design a payroll module that manages the salary details of employees.
- Who do you think has access to an employee's salary details?

Think and Tell

- Who do you think should know salary details?
  - Only the employee
  - All the members of the company
  - The employee and a few other employees of the company
- Which is the correct answer according to you?

Introduction to Encapsulation and Data Abstraction

Menu

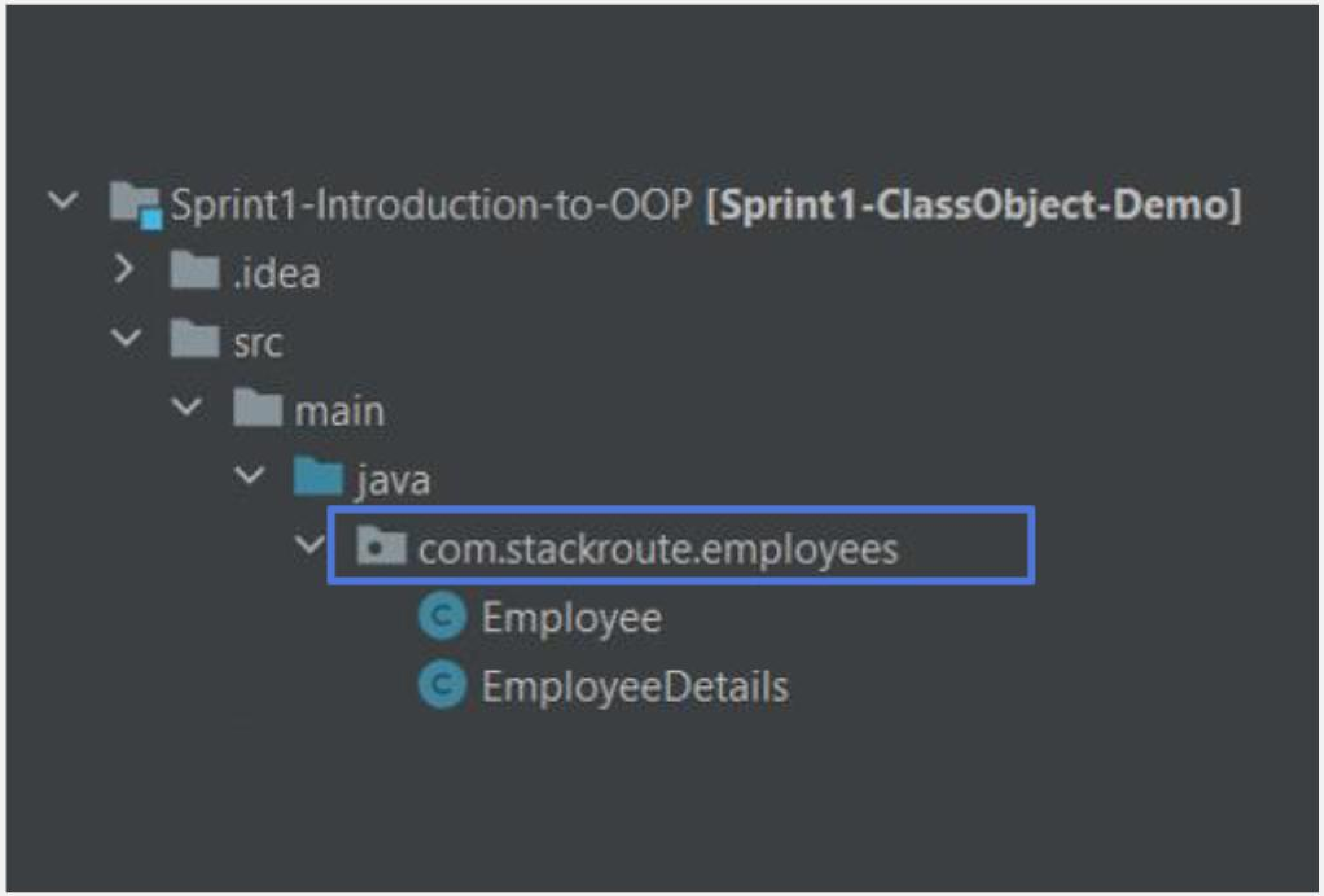
00:09 / 00:00:44

09/ 44

Learning Objectives

- Organize classes using packages
- Explore the pillars of OOP
- Explain encapsulation
- Use access specifiers
- Implement encapsulation
- Explain abstraction

Organize Classes Using Packages



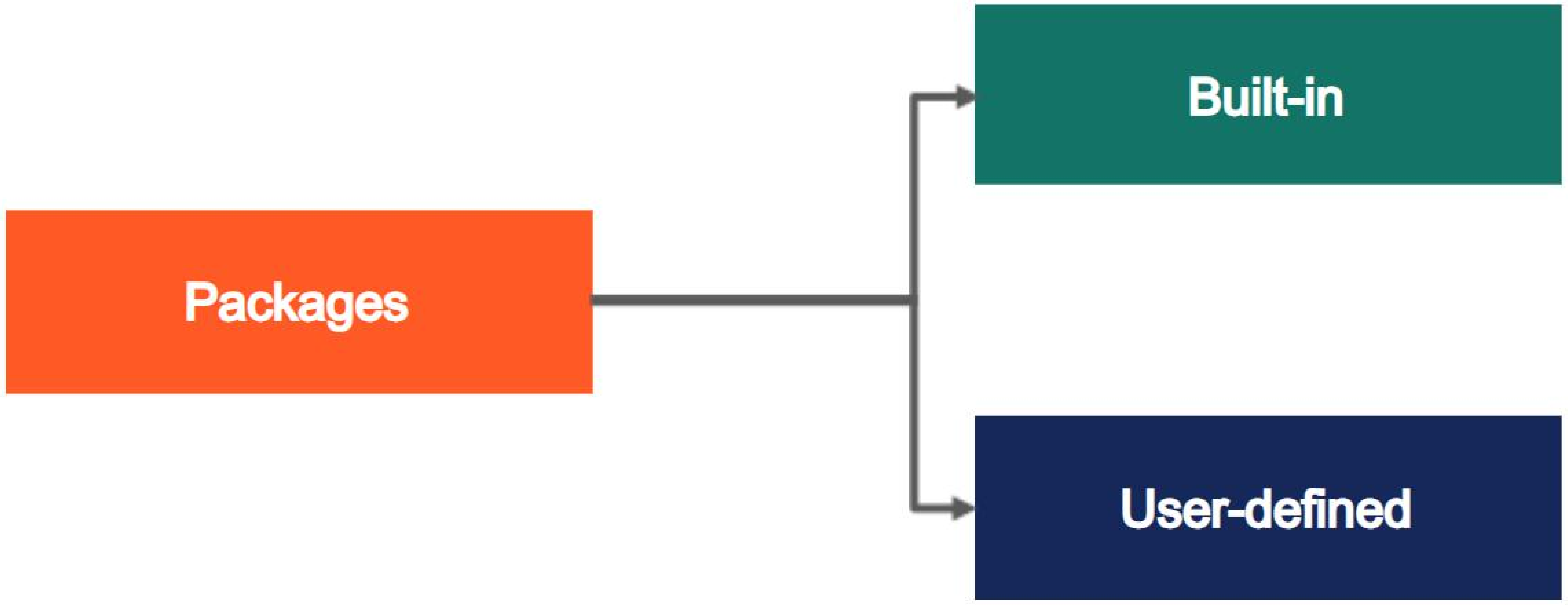
```
Sprint1-Introduction-to-OOP [Sprint1-ClassObject-Demo]
  .idea
  src
    main
      java
        com.stackroute.employees
          Employee
          EmployeeDetails
```

Introduction to Package

What Is a package?

- A package is a namespace that organizes a set of related classes.
- Packages are containers for classes.
- The `com.stackroute.employees` package contains the `Employee` and `EmployeeDetails classes.`
- One project can have multiple packages that have the Java classes organized under them.

Types of Packages



Packages can be of two types:

- Built-in packages —— provided by the Java libraries
- User-defined packages —— created by the programmer

Built-in Packages

- The built-in packages are present in the Java class libraries.
- The `java.lang` and `java.util` packages are the most popular packages whose classes are used extensively.
- The `System` and `Exception` classes are part of the `java.lang` package.

Quick Check

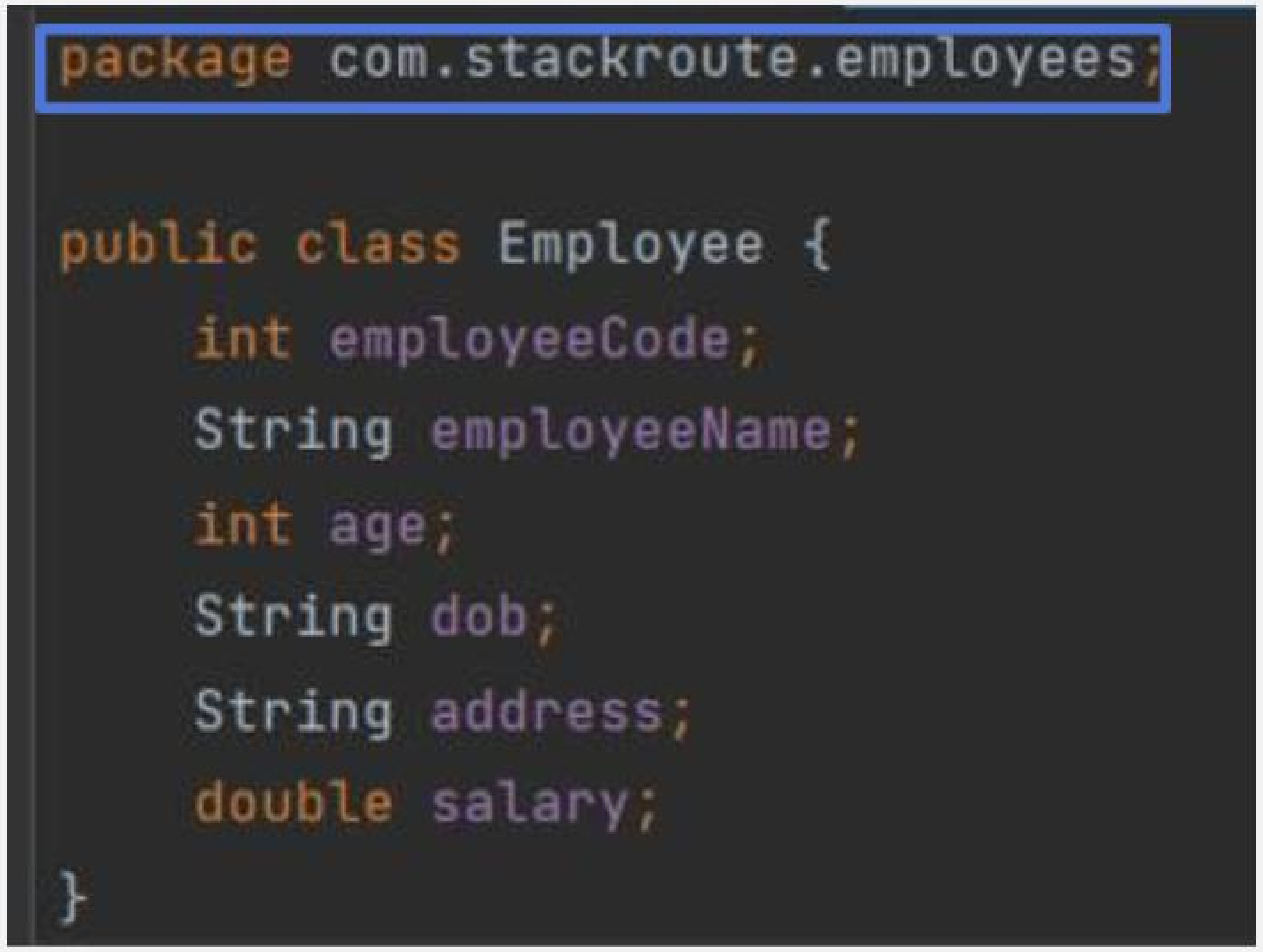
The `Math` class is present in the ___________ package.

1. `java.sql;`
2. `java.net;`
3. `java.lang;`
4. `java.util;`

Quick Check: Solution

The `Math` class is present in the ___________ package.

1. `java.sql;`
2. `java.net;`
3. **`java.lang;`**
4. `java.util;`



```java
package com.stackroute.employees;

public class Employee {
    int employeeCode;
    String employeeName;
    int age;
    String dob;
    String address;
    double salary;
}
```

The class `Employee` **belongs to the** `com.stackroute.employees` **package.**

User-Defined Packages

- User-defined packages can be created by the user to organize classes.
- IDEs provide a structure to the Java project and automatically create a package structure.
- The first line of the class defines the package in which the class is present.
- `package` is a keyword used to specify which package the class belongs to.
- The package name must be lowercase.

Importing Packages

- The classes inside the built-in and user-defined packages can be used in other Java programs outside the same package.
- The best example of using a built-in package is utilizing the `Scanner` object.
- `Scanner` is a class that is part of the `java.util` package.
- To use the Scanner class from the `java.util` package, we use the import statement.

```
import java.util.Scanner;
```

- The user-defined packages can also be imported in the same manner.

```
import com.stackroute.employees;
```

- If all the classes inside the package need to be utilized in a class, we can use a wild card character '*'.

```
import java.util.*;
```

Quick Check

Packages are represented and stored by Java as ____________ in the file system.

1. files
2. directories
3. applications
4. data

Quick Check: Solution

Packages are represented and stored by Java as ____________ in the file system.

1. files
2. directories
3. applications
4. data

Creating a Book Class

Create a `Book` class with attributes such as the ISBN, title, and year of publication inside a package called "library" with a method to display the details of the `Book` class. Create an implementation class called `BookImpl` inside the library package. Call the methods of the `Book` class in the main method of the `BookImpl` class.

Click here for the solution.
Demonstrate the program using the IntelliJ IDE.

DEMO

Explore the Pillars of OOP

The Pillars of OOP

- Object-oriented programming (OOP) is built on a few basic principles called the pillars of OOP.

Encapsulation

Abstraction

Inheritance

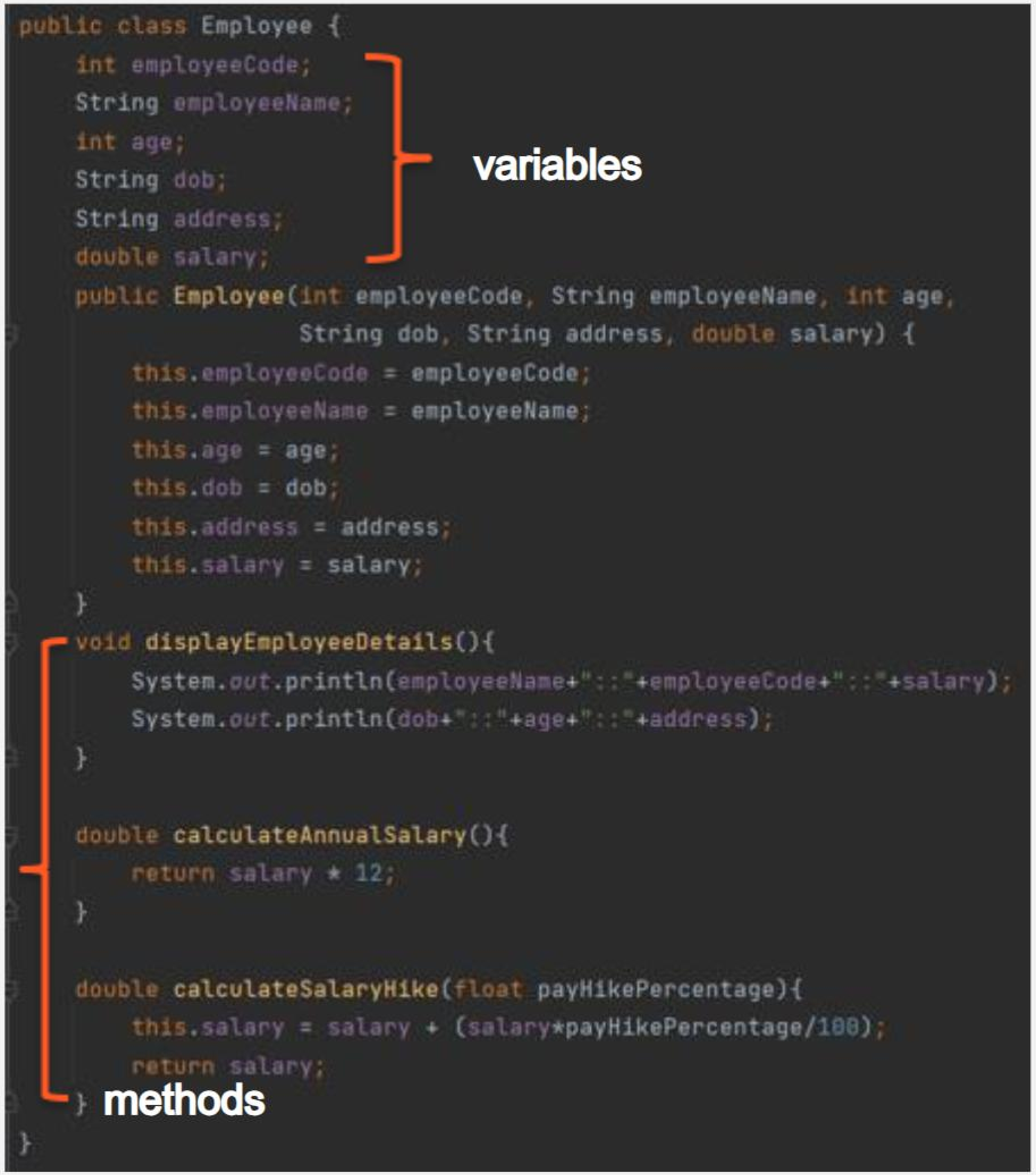
Polymorphism

- Note: We will explore each of these in detail in this course.

Explain Encapsulation

What Is Encapsulation?

- Encapsulation is the wrapping of data and code.
- The data is the variables declared in the class.
- The code is the methods that work on the data.
- A class consists of variables and methods that work on those variables.



```java
public class Employee {
    int employeeCode;
    String employeeName;
    int age;
    String dob;
    String address;
    double salary;
    public Employee(int employeeCode, String employeeName, int age,
                    String dob, String address, double salary) {
        this.employeeCode = employeeCode;
        this.employeeName = employeeName;
        this.age = age;
        this.dob = dob;
        this.address = address;
        this.salary = salary;
    }
    void displayEmployeeDetails(){
        System.out.println(employeeName+"::"+employeeCode+"::"+salary);
        System.out.println(dob+"::"+age+"::"+address);
    }

    double calculateAnnualSalary(){
        return salary * 12;
    }

    double calculateSalaryHike(float payHikePercentage){
        this.salary = salary + (salary*payHikePercentage/100);
        return salary;
    }
}
```

Encapsulation in Java

- Encapsulation is achieved in Java using:
  - Classes: Classes wrap the variables and methods into one single unit accessed only by creating an object.
  - Access specifiers: They restrict access to the variables in a class at multiple levels.
  - Setter and getter methods: These methods prevent variables from misuse or unwanted changes by other objects.

Use Access Specifiers

Access Specifiers

- Java access specifiers also known as visibility specifiers. They regulate access to classes, variables, and methods in Java.
- These specifiers determine whether a variable or method in a class can be used or invoked by another class in the same package or in another package.
- Access specification can be provided to the classes and the members of the class.
- Java uses the keywords `private`, `public`, and `protected`.
- If none of the keywords are specified, then it is considered default access.

Access Specifiers

- The table below specifies how the access specifiers facilitate a class, variable, or a method to be used in classes within the same package or outside the package.

	Default	Private	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Quick Check

How can you declare a class `Room` **in a package** `building` **so that it is not visible outside the package** `building` **?**

1. `package building; public class Room{}`
2. `package building; protected class Room{}`
3. `package building; class Room{}`
4. `package building; private class Room{}`

Quick Check: Solution

How can you declare a class `Room` **in a package** `building` **so that it is not visible outside the package** `building` **?**

1. `package building; public class Room{}`

2. `package building; protected class Room{}`

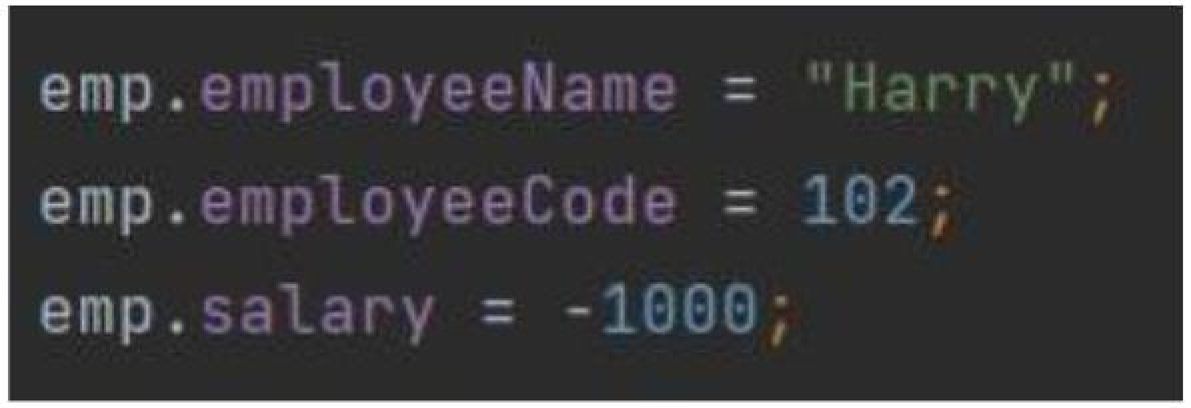
3. **`package building; class Room{}`**

4. `package building; private class Room{}`

Implement Encapsulation

Implementing Encapsulation

- Encapsulation of data and code in a class can be achieved using access specifiers.
- In the class `Employee,` all the variables and methods have default access, which means all the values can be accessed by all the classes in the same package directly by using the '.' operator as shown below.



```
emp.employeeName = "Harry";
emp.employeeCode = 102;
emp.salary = -1000;
```

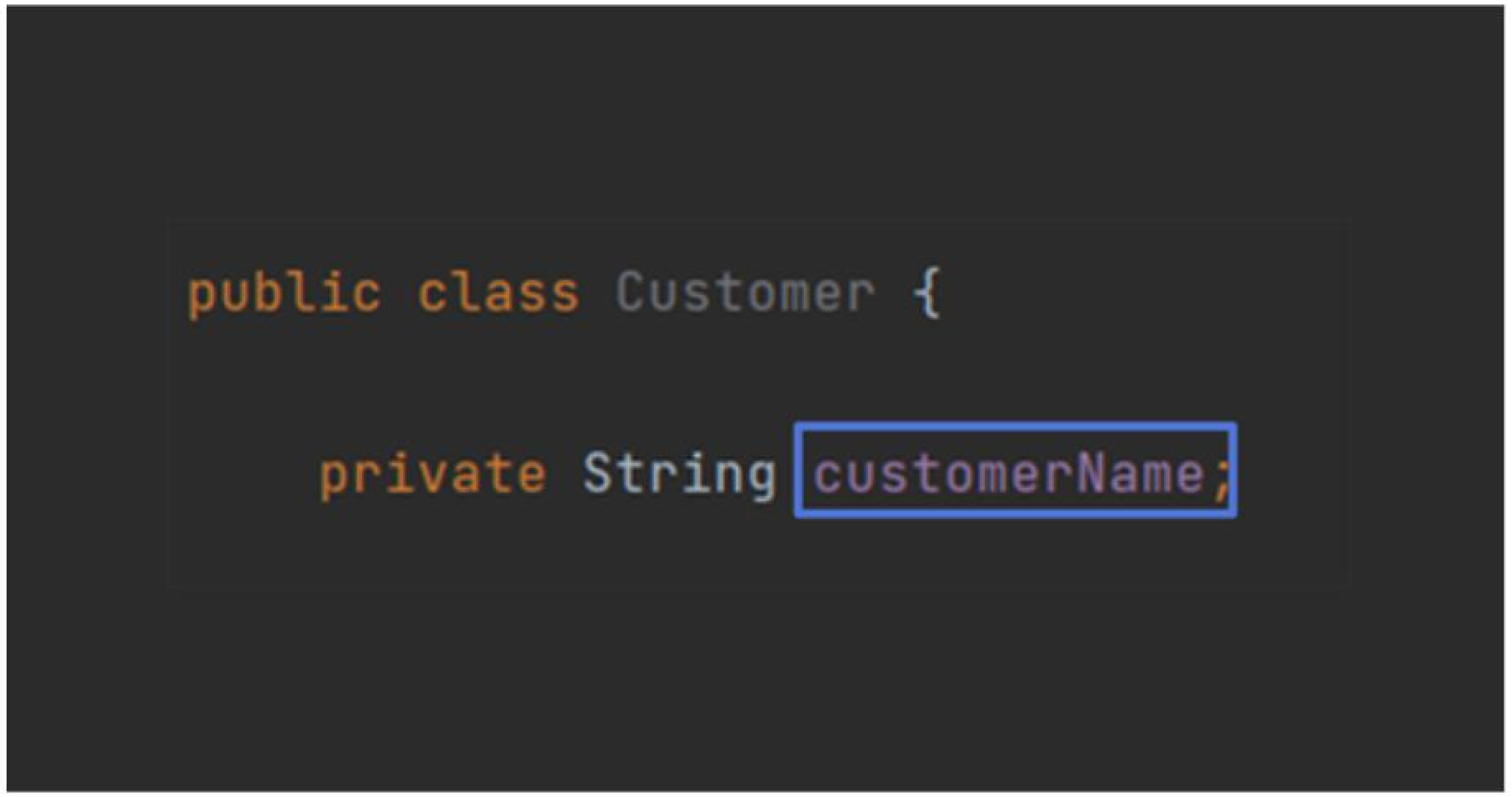
- Can other classes directly access the variables and alter their values?
  - Yes, other classes can alter the variable values.
- What will happen if the value of the `salary` variable is set to a negative value?
  - This will be a problem since salary cannot be a negative value.

Implementing Encapsulation – Accessor/Mutator Methods

- A well-encapsulated class can be created by generating accessor and mutator or getter and setter methods for the variables.
- The variables are made private, and access to the variables is provided through getter and setter methods.
- The value of the variable can be fetched with the getter or accessor methods.
- The value of the variable can be modified with the setter or mutator methods.
- If the programmer wishes to specify that no other class should modify the variable value, the setter method can be made `private`.
- If a variable cannot be assigned a negative value, a check can be made inside the setter method, before the value is assigned.

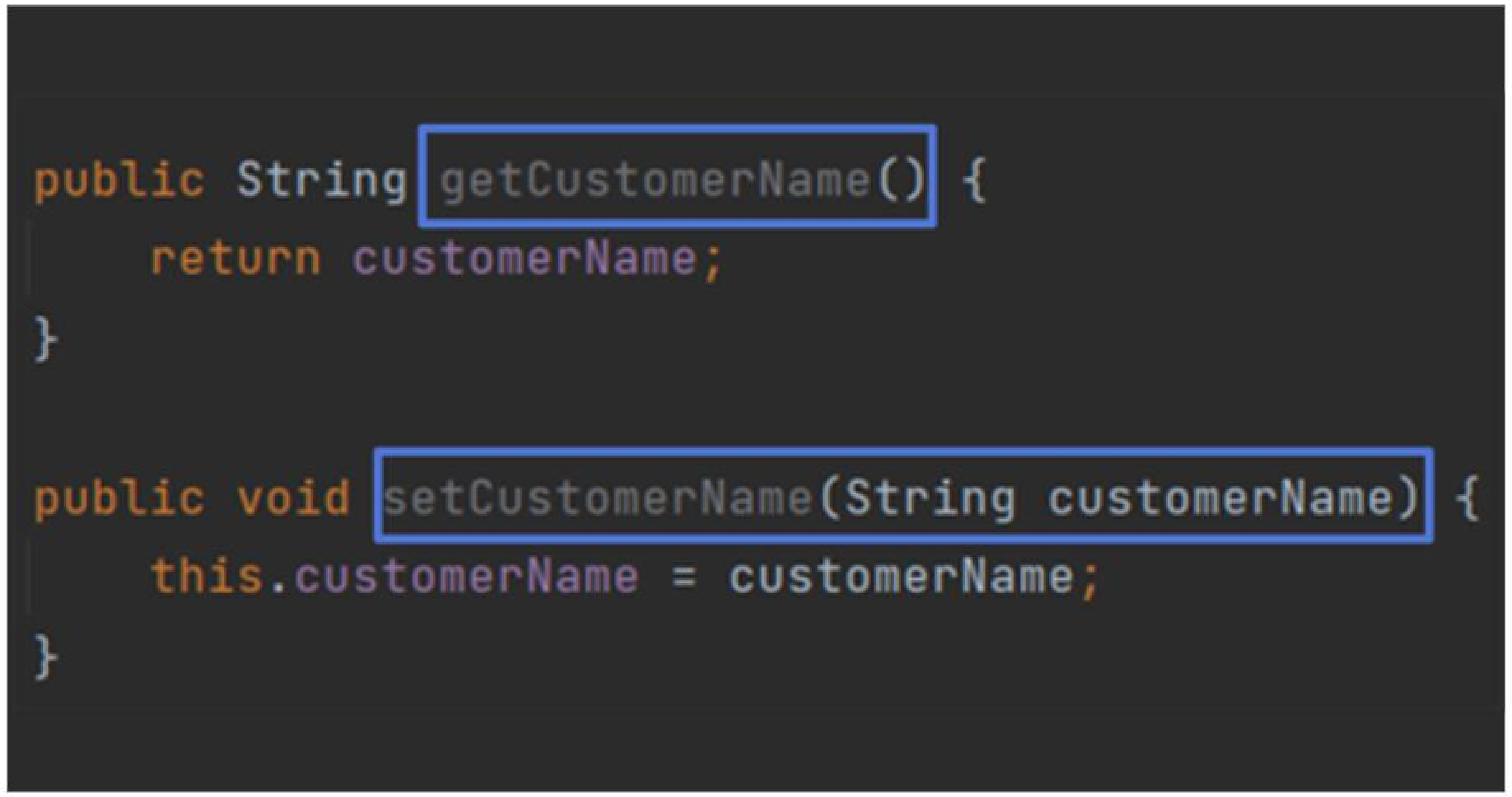
Naming Conventions for Getter/Setter Methods

- Getter and setter method names are composed of the words *get* or *set*, respectively, plus the property name, with the first character of each word capitalized.
- A regular getter method has no parameters but returns a value of the property's type.
- A setter method has a single parameter of the property's type and has a void return type.
- Consider a class `Customer` with the attribute Customer Name. The setter and getter must be as follows:



```
public class Customer {

    private String customerName;
```

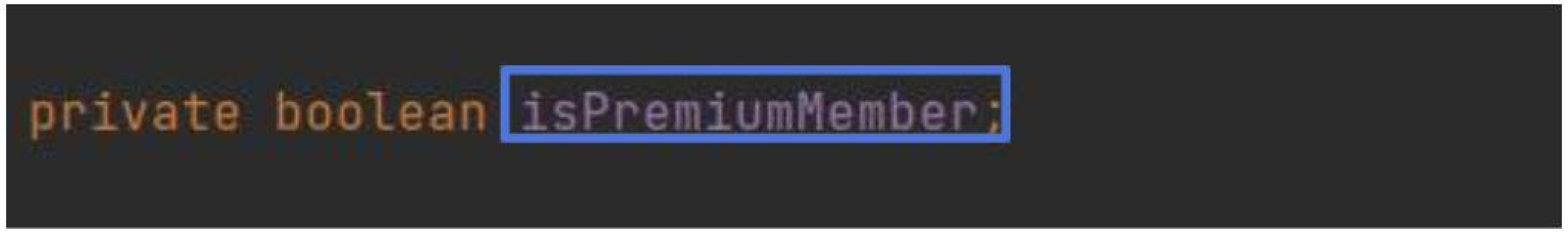


```
public String getCustomerName() {
    return customerName;
}

public void setCustomerName(String customerName) {
    this.customerName = customerName;
}
```

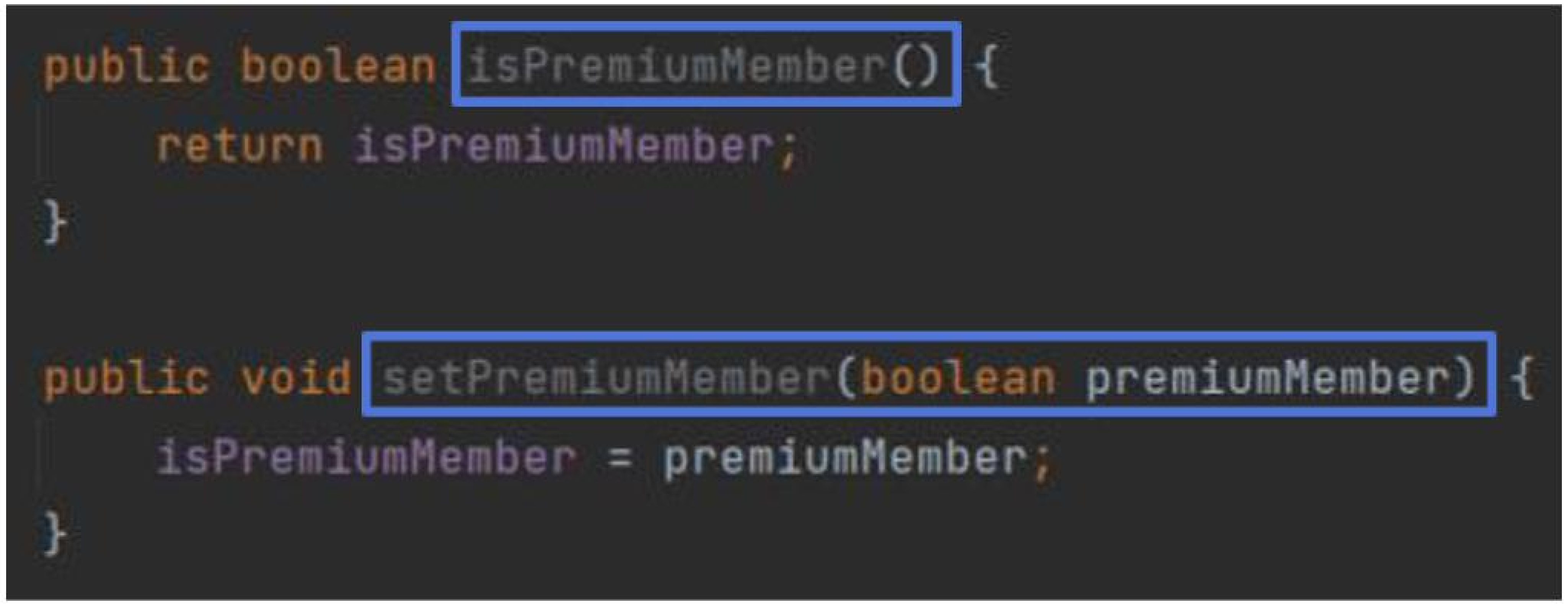
Naming Conventions for Getter/Setter Methods

- If a variable is declared as `boolean` that represents if the customer is a premium member, the getter and setter must be as follows:



```java
private boolean isPremiumMember;
```

- Note that the getter method does not have a "get" prefixed to the variable name.
- The setter method is prefixed with "set" as per convention.



```java
public boolean isPremiumMember() {
    return isPremiumMember;
}

public void setPremiumMember(boolean premiumMember) {
    isPremiumMember = premiumMember;
}
```

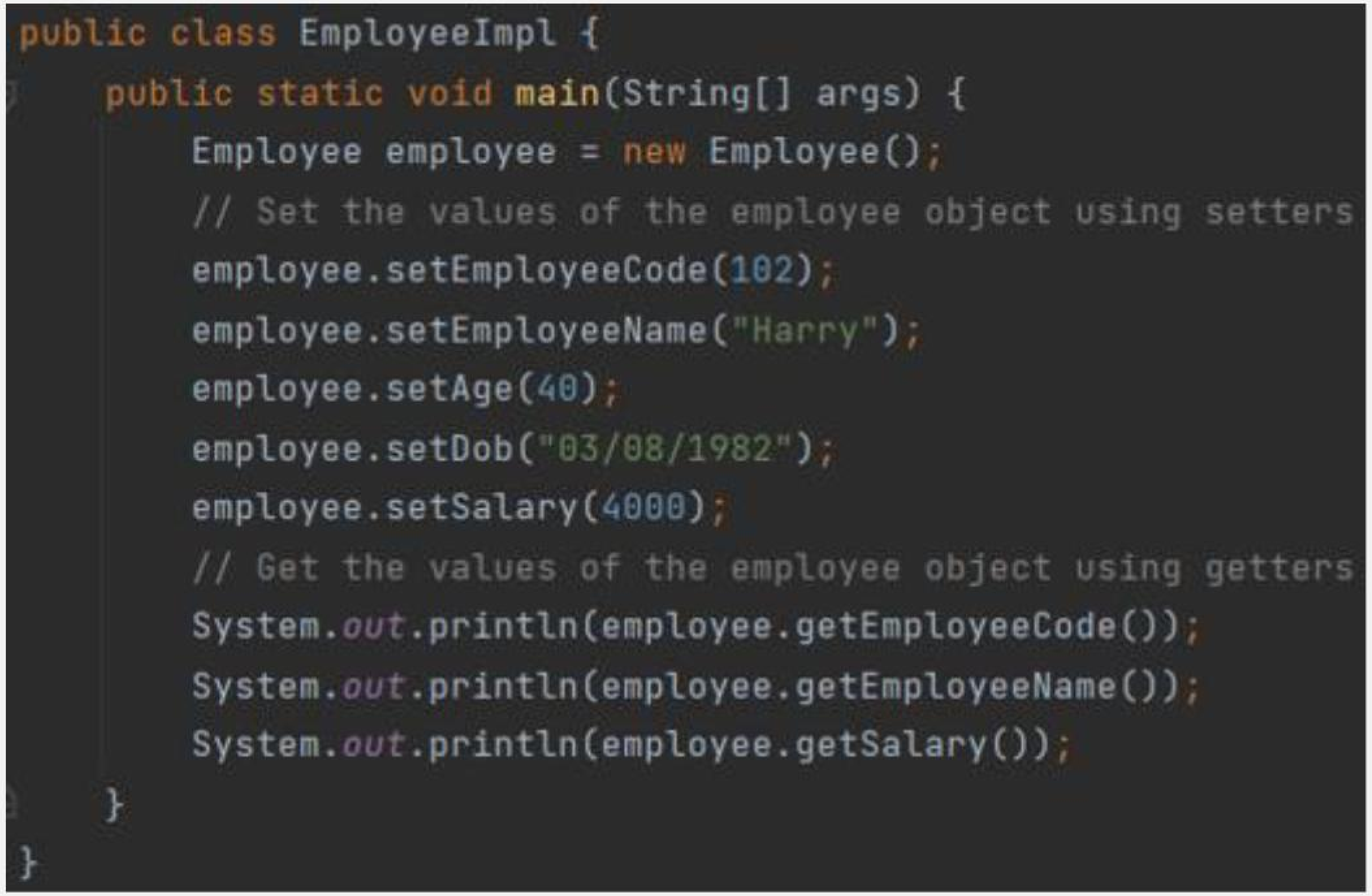
A Well-Encapsulated Class

The `Employee` class with accessor and mutator methods

```java
public class Employee {
    private int employeeCode;
    private String employeeName;
    private int age;
    private String dob;
    private String address;
    private double salary;
    public int getEmployeeCode()
    { return employeeCode;  }
    public void setEmployeeCode(int employeeCode)
    { this.employeeCode = employeeCode; }
    public String getEmployeeName()
    { return employeeName; }
    public void setEmployeeName(String employeeName)
    { this.employeeName = employeeName;}
    public int getAge()
    { return age; }
```

```java
public void setAge(int age)
{this.age = age;}
public String getDob()
{ return dob; }
public void setDob(String dob)
{ this.dob = dob; }
public String getAddress() {
    return address; }
public void setAddress(String address) {
    this.address = address; }
public double getSalary() { return salary;   }
public void setSalary(double salary) {
    this.salary = salary; }
```

The Implementation Class



```
public class EmployeeImpl {
    public static void main(String[] args) {
        Employee employee = new Employee();
        // Set the values of the employee object using setters
        employee.setEmployeeCode(102);
        employee.setEmployeeName("Harry");
        employee.setAge(40);
        employee.setDob("03/08/1982");
        employee.setSalary(4000);
        // Get the values of the employee object using getters
        System.out.println(employee.getEmployeeCode());
        System.out.println(employee.getEmployeeName());
        System.out.println(employee.getSalary());
    }
}
```

- **The `EmployeeImpl` class is in the same package as the `Employee` class.**
- **The values for the variables can be modified or fetched using getter and setter methods instead of directly accessing the variable using the '.' operator.**
- **The values can be initialized using the setter methods without having to declare a parameterized constructor.**

Initializing Values to Variables

- Why use getter and setter methods for initializing variables when constructors do the job efficiently?
  - When an object is instantiated, its constructor is called, and the private variables are available to the constructor to be set. Internally to the class all variables are directly accessible.
  - If other classes need the data, they are not allowed to access it. The class itself receives commands to alter its data from other codes.
  - So, in order to avoid corruption from outside, setter/getter functions are created that provide a public interface to the outside code.
  - Getters/setters protect the variables from badly formatted data and prevent programmers from accessing the values directly.

Constructor vs. Setter Methods

Constructor	Setter
Constructor is a one-time initializer of values of variables at the time of object creation.	Setters can be used to set up values later after the object is created.
Constructors are by default available to a class at the time of object creation.	Setters are optional.

- A class can have a constructor and getter/setter methods.
- To provide initial values to the object, a constructor can be used. To set up values for individual attributes, setters can be used.
- They are purely design decisions.

Restructuring the College Library

Sam, a college librarian, struggles to get his books organized. To structure the library and make the books more accessible to the students, the college authorities have decided to create a library management system. This system will provide an online platform to search and access a variety of books and materials. To do so, Sam must create a `Book` class that will help the system organize the books. Help Sam create the class and capture basic attributes, such as the title, year of publication, ISBN, author name, etc.

Create an implementation class called `BookImpl` and set values to the attributes using the setter methods. Display the details of the books.
Click here for the solution.
Demonstrate the program using the IntelliJ IDE.

DEMO

Menu

Explain Abstraction

What Is Abstraction?

- Abstraction is about hiding unwanted details and exposing only the most essential information.
- When the object data is not visible to outside its own class, it creates data abstraction.
- Both the variables and methods of a class can be abstracted.
- In Java, abstraction is achieved using abstract classes and interfaces, which will be discussed in detail later on.