IN1007 Programing in Java

Lecture 4: Introduction to Classes and Objects

Announcements

- **First coursework available TODAY** → deadline 17th Nov (see details on Moodle)

- **Mid term survey:** Take 5 minutes to tell me about your experience with the module so far

Today's Lecture

- Concepts of OOP
  - What is an **object**?
  - **States** and **behaviours** of an object
  - **Encapsulation**

- **Defining** classes
- **Declaring** and initialising objects
- **Accessing** the state and behaviour of an object

Object Oriented Programming

Source: [What is Object-Oriented Programming? | Coding for Kids | Kodable - YouTube]

Concepts of OOP: Object

What is an object?

→ The ***things*** you think about first when designing a "world": characters, vehicles, etc.

Example: Dog

Image source: Easy Cartoon Dog Tutorial - FeltMagnet

State:

```
name = "Lilo";
age = 2;
breed = "Golden";
…
```

Behaviour:

```
Wag_tail()
bark()
Fetch()
…
```

Concepts of OOP: Object

State of an object is represented by the object's variables, also known as attributes or fields

Behaviours of an object are represented by methods

Example: Dog

Image source: Easy Cartoon Dog Tutorial - FeltMagnet

State:

```
name = "Lilo";
age = 2;
breed = "Shih Tzu";
…
```

Example: Second dog?

Image source: Easy Cartoon Dog Tutorial - FeltMagnet

State:

```
name = ?
age = ?
breed = ?
…
```

Principles of OOP: Encapsulation

Data encapsulation

- Hiding the internal state (i.e. ***fields***) of an object from the outside "world" so that no unintentional changes can be applied

- Changes to the state of an object should only be performed through the object's methods

State:

```
name = ?
age = ?
breed = ?
…
```

```
setName()
setAge()
setBreed()
```

Today's Lecture

- Concepts of OOP
  - What is an **object**?
  - **States** and **behaviours** of an object
  - **Encapsulation**

- **Defining** classes
- **Declaring** and initialising objects
- **Accessing** the state and behaviour of an object

Defining a Class

Before creating an object, there needs to be a definition of:

- What is this type of object expected to be? What possible states can it be in?
- How is this type of object expected to behave?

A class: is a prototype of an object that lists the ***fields*** and ***methods*** which define possible states and behaviours

Code for a class

```
class MyClass {
    // field and
    // method declarations
}
```

Exercise 1: create a class named Student
What fields and methods should be defined?

Discuss!

Exercise 1: The *Student* class

State variables:

* studentName

* studentNumber

** other…?*

Methods:

* access to class state

** other…?*

```java
public class Student {
    String studentName;
    int studentNumber;

    void updateName(String newName){
        studentName = newName;
    }
    void updateNumber(int newNumber){
        studentNumber = newNumber;
    }
}
```

Today's Lecture

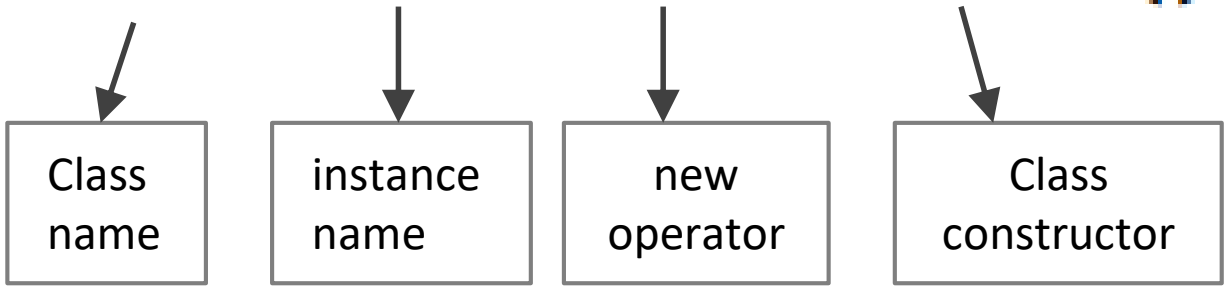
- Concepts of OOP
  - What is an **object**?
  - **States** and **behaviours** of an object
  - **Encapsulation**

- **Defining** classes
- **Declaring** and initialising objects
- **Accessing** the state and behaviour of an object

Instantiating a Class

Instantiating a class = creating an object = creating a class instance

```
Student s = new Student();
```



Creating a *Student* object

Student.java Main.java

```java
public class Main {
    public static void main(String[] args){
        Student s = new Student();
        System.out.println(s.studentName);
    }
}
```

Output: *null*

Calling a member method

```java
public class Main {
    public static void main(String[] args){
        Student s = new Student();
        s.updateName("Samy");
        System.out.println(s.studentName);
    }
}
```

Output: *Samy*

Calling a member method

Class instance

```
public class Main {
    public static void main(String[] args){
        Student s = new Student();
        s.updateName("Samy");
        System.out.println(s.studentName);
    }
}
```

Output: *Samy*

Class Constructor

- A special type of method in Java
- Enables us to create an object that instantiates a class
- Constructors are typically the first method(s) we define in a class
- Hold the same name as the class

Exercise 2: Write a constructor

See if you can create **two** constructors for the Student class

Hint: remember method overloading

```java
public class Student {
    String studentName;
    int studentNumber;

    void updateName(String newName){
        studentName = newName;
    }
    void updateNumber(int newNumber){
        studentNumber = newNumber;
    }
}
```

Exercise 2 solution:

Possible constructors

```java
public class Student {
    String studentName;
    int studentNumber;

    Student(int newNumber) {
        studentNumber = newNumber;
        studentName = "";
    }

    Student(int newNumber, String newName) {
        studentNumber = newNumber;
        studentName = newName;
    }
}
```

Exercise 3.a: Write a class called Film

Write a class Film, with the following fields:

- a field `title` of type String,
- a field `releaseYear` of type int,
- a field `duration` of type double,
- a field `soldTickets` of type int.

Exercise 3.a Solution: class Film

```
class Film {
    String title;
    int releaseYear;
    double duration;
    int soldTickets;
}
```

Exercise 3.b:

- Write a constructor for the class Film

- Write a method called `sellTickets()` that accepts a number of sold tickets and updates the corresponding field

Exercise 3.b solution: Part A

```java
public class Film {
    String title;
    int releaseYear;
    double duration;
    int soldTickets;

    Film(String title, int releaseYear, double duration){
        this.title = title;
        this.releaseYear = releaseYear;
        this.duration = duration;
        this.soldTickets = 0;
    }
}
```

The `this` keyword

- The `this` keyword refers to the current object in a method or constructor.

- The most common use of the `this` keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter).

Try it yourself: https://tinyurl.com/ur87u26e

Exercise 3.b solution: Part B

```java
public void sellTickets(int numTickets){
    soldTickets += numTickets;
}
```

- ✓ Try adding `this` keyword
- ✓ No difference in results. *Why?*

Creating and accessing objects

```
1  public class Main {
2      public static void main(String[] args){
3          Student s = new Student();
4          s.updateName("Samy");
5          System.out.println(s.stu
6      }
7  }
8
```

Cannot resolve constructor 'Student()'

Create constructor Alt+Shift+Enter More actions... Alt+Enter

Candidates for new **Student**() are:
Student(int newNumber)
Student(int newNumber, String newName)
week7

After creating a constructor for class Student, I got this error. *Why?*

Today's Lecture

- Concepts of OOP
  - What is an **object**?
  - **States** and **behaviours** of an object
  - **Encapsulation**

- **Defining** classes
- **Declaring** and initialising objects
- **Accessing** the state and behaviour of an object

Exercise 4: Accessing an object of class Film

```
public static void main(String[] args){
    Film myfilm = new Film("The Sound of Music", 1965, 2.75 );
    System.out.print(myfilm.duration);
}
```

Does the `Film` class uphold the concept of data hiding or encapsulation?

Discuss!

Encapsulation again!

- Only methods of the class can access its data fields
- A way for hiding implementation details of a class from outside access
- In Java, encapsulation is achieved by declaring fields as `private`

Image source What is Encapsulation in Java and How to Implement It (simplilearn.com):

Apply these changes to the class Film

```
public class Film {
    String title;
    private int releaseYear;
    private double duration;
    private int soldTickets;
```

This is called an
access modifier

Now your main method shows this error. *How can we fix this?*

Tool

Main

Student.java | Film.java | Main.java

```
public class Main {
    public static void main(String[] args){
        Film myfilm = new Film("The Sound of Music", 1965, 2.75 );
        System.out.print(myfilm.duration);
    }
}
```

1

'duration' has private access in 'Film'

Make 'Film.duration' package-private Alt+Shift+Enter More actions... Alt+Enter

Film

private double duration

week7

Encapsulation again!

- **Only methods of the class can access its data fields**
- A way for hiding implementation details of a class from outside access
- In Java, encapsulation is achieved by declaring fields as `private`

So, let's create a method!

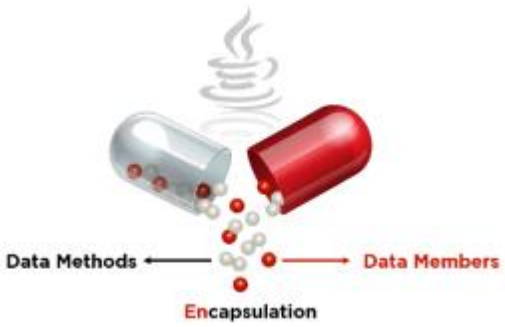


Image source What is Encapsulation in Java and How to Implement It (simplilearn.com):

Exercise 4 Solution

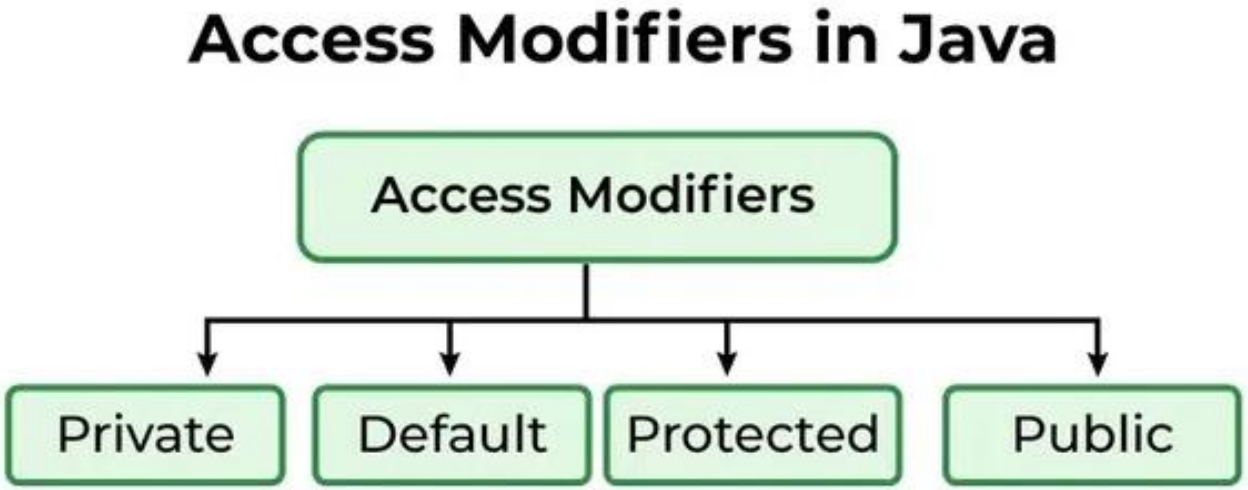
Changes to Film.java:

```java
public double getDuration(){
    return duration;
}
```

Changes to Main.java:

```java
public class Main {
    public static void main(String[] args){
        Film myfilm = new Film("The Sound of Music", 1965, 2.75 );
        System.out.print(myfilm.getDuration());
    }
}
```

Introduction to Access Modifiers in Java



Read about Java access modifiers at:
https://www.geeksforgeeks.org/access-modifiers-java/

Recap: How to access a class member?

To access from inside the class:

- You may use the variable name directly
- You may use the `this` keyword to avoid confusion with method parameters that share the same name

To access a field or method of an object (outside of class definition):

- Use the notation `objectName.memberName`
- Use the correct access modifier (depends on where in the code you are trying to access. Use public to access anywhere).

Today's Lecture

- Concepts of OOP
  - What is an **object**?
  - **States** and **behaviours** of an object
  - **Encapsulation**

- **Defining** classes
- **Declaring** and initialising objects
- **Accessing** the state and behaviour of an object