Féidearthachtaí as Cuimse
Infinite Possibilities

Getting Deeper with OOP: Classes and Objects

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN

W2-2025, Object-Oriented Programming, Week 2

Objectives

- Creating a Java Class
    - The Anatomy of a Class

Start with Java class

- Basic class definition

```
class className
 {
     // attributes
          …
     // methods
          …
}
```

Defining Classes:

Define a class using a **class** keyword followed by the class name.

Define attributes and methods with the class.

Attributes and methods together are called “class members”

Car Class (on BrightSpace)

- Note:
  - Class name (with capital letters)
  - Access modifiers
  - Data types *(static typing in java)*
  - Methods (if any)
  - Constructor (missing at the moment – so uses default)
  - Comments

```java
package Friday;

//import java.lang.String //why do I not import

public class Car {

    // Attributes (fields)
    private String model;        // declaration only
    private int year = 2020;     // with initial value
    public double price;         // accessible outside


    // Method with no return
    public void startEngine() {
        System.out.println("Engine started.");
    }

    // Method with return value
    public int getYear() {
        return year;   // returns attribute
    }

    // Method with parameters
    public void setModel(String newModel) {
        model = newModel;
    }
}
```

Attributes

```
public class Car {

    // Attributes (fields)

    private String model;

    private int year;

    public double price;

}
```

- **accessModifier** → public, private, protected
- **dataType** → type of the variable (int, String, etc.)
- **attributeName** → variable name (camelCase)

Attribute Types (Can be)

Primitive Types (basic built-in types)

```
int age;

double price;

boolean isAvailable;

char grade;
```

Reference Types

```
String name;            // object (String class)

Car anotherCar;         // reference object

int[] scores;           // array

List<String> authors;   // collection
```

String, Car, List<String> are data types (classes).
name, anotherCar, scores, authors are variables (attributes) of those data types.

Reference types store memory references (pointers to objects).

Methods (overview covered in separate slide deck)

```
accessModifier returnType methodName(parameterList) {

    // method body

    return value;    // if not void

}
```

```java
// Method with no return
public void startEngine() {
    System.out.println("Engine started.");
}

// Method with return value
public int getYear() {
    return year;    // returns attribute
}

// Method with parameters
public void setModel(String newModel) {
    model = newModel;
}
```

Constructors

Special method (in the class) that **initializes attributes** when an object is created

Purpose

Set **initial values** for object fields. (hence its name)

Rules

Name must **match the class name** exactly.

No return type, not even void.

Automatically called when you use new.

Type of Java Constructor

- Default (no-arg constructor)
- Parametrized constructor

```
1  package w2;
2
3  public class Book {
4
5      int m_isbn;
6      String m_title;
7      String m_author;
8
9      public Book()
10     {
11
12     }
13
14     public Book(int ibsn, String title)
15     {
16
17         m_isbn = ibsn;
18         m_title = title;
19     }
20
21     //two things I can do
22
23     public static void main(String[] args) {
24          // Creating a person object
25         Book testbook = new Book();
26         Book javabook = new Book (1234567,"TUD Learning Java");
27
28     }
29
30 }
```

Note: In Java, parameter names are arbitrary identifiers chosen by the programmer — they can be any valid variable name.

When you create an object you pass values into the constructor- if parametrized

Constructors

- Used to create objects for a class
  - May need more than one in a class (why?)
  - e.g.

```
Book b1 = new Book();
Book b2 = new Book("Dune");
Book b3 = new Book("Dune", 41);
```

  - Code: multiple constructors

```
// Constructor
Book(String t, String a) {
    title = t;
    author = a;
    isBorrowed = false; // all new books start as not borrowed
}
```

```
Car myCar = new Car();
```

In Java, when you use new, it runs a constructor to build the object.
new creates it, the constructor sets it up.

Creating Objects and Instantiation

Instantiation:

The process of creating an instance (object) from a class.

Creates a unique copy of the class blueprint.

Creating Objects:

Use the new keyword followed by the class name and parentheses.

Objects can be created with or without constructor arguments.

- **Accessing Attributes**:
  - Use dot notation with methods (getters/setters) to interact with fields.
  - Direct field access is possible if the field is public (but not recommended). (I do this initially in demo 1 for display only)
- **Creating Objects Dynamically**:
  - Objects can be created during program execution inside loops or based on conditions
  - This adds flexibility and adaptability to your code.

A Special Keyword: this

```java
package week2Demo4;

public class Book {
    private String title;
    private String author;
    private boolean borrowed;

    public Book(String title, String author) {
        this.title = title;
        this.author = author;
        this.borrowed = false;
    }
```

- In Java, **this** is like a way for an object to refer to itself.
- It's like saying "I" when talking about yourself.
- When you use **this** in a method inside a class, you're saying:
- *"I want to do something with my own data or behaviour."*
- It helps Java know which object's fields or methods you're referring to, especially when there are **multiple objects of the same class** or when method parameters have the same name as fields.

this Cont’d

When you create a class in Java, you’re defining a blueprint for creating objects.

These objects will have attributes (fields) and methods that operate on that data.

Inside the methods of a class, you often need to refer to the attributes and methods of the particular object you’re working with. This is where **this** comes in .**this** is a reference to the current object.

It helps Java differentiate between the attributes and methods of one object and those of another when you have multiple objects of the same class.

True or False?

> The constructor in Java is automatically called when you create an object from a class using the new keyword.

Java classes

- To run a java program, need **a special method** called the `main` method to tell the program where to start executing
- Put the “main” method in its own class OR as a method in an existing class (if just using one class).

```
public static void main (String [] args)
{
   // e.g.          ..
}
```

True or False?

> In Java, objects are instances of classes, and you create objects by calling a class's constructor using the new keyword.

Accessing Attributes and Methods

- **Accessing Attributes:**
  - Use dot notation (object.attribute) to access object attributes.
  - Attributes store data specific to each object.
  - Attributes can be read or modified.
- **Accessing Methods:**
  - Use dot notation (object.method()) to call object methods.
  - Methods define behaviour and actions of objects.

Benefits:

- Objects encapsulate data and behaviour.
- Dot notation provides a clear way to interact with object attributes and methods.
- Enables effective manipulation and communication with objects.

Example of Accessing Attributes and Methods

```java
package week1Demo1;

class Person {
    String name;
    int age;

    // Constructor
    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method
    void greet() {
        System.out.println("Hello, my name is " + name + ".");
    }
}

// Main class to run the program
public class Main {
    public static void main(String[] args) {
        // Creating a person object
        Person person = new Person("Alice", 25);

        // Accessing attributes
        System.out.println(person.age);  // Output: 25

        // Calling a method
        person.greet();  // Output: Hello, my name is Alice.
    }
}
```

True or False?

> In Object-Oriented Programming (OOP), objects only contain data (attributes) and cannot have associated actions (methods).

Anatomy of a Class

Class Declaration

class + **PascalCase** name.

Body in { }.

Fields (Attributes)

Declared inside class.

Have a type (String, int).

Usually private (encapsulation).

Methods

Define behaviour; have return type or void.

Usually public; can be private.

Constructor

Same name as class, no return type.

Called with new.

Creating & Using Objects

Book b = new Book("Dune", 412);

Access: b.read();, b.getPages();

Objects Represent Real-World Entities:

Classes and objects mirror real-world entities and their properties/actions.

Key Takeaways

- Lots of new terms today; it’s going to be less in the following weeks!
- Transition from procedural to OOP introduces a paradigm shift.
- We wrote our first classes in Java.
- OOP emphasizes classes, objects, attributes, and methods.