Object-Oriented Programming

CS 101

Algorithms and Programming 1

suggested reading:

Chapter 6

Big Java Late Objects

Adapted from CS101 Lecture Notes

Software Paradigm

- Paradigm - guides the way that developers view a given problem and organize the solution.
- Choosing the right development process is important:
- Right process?
    - Faster development,
    - reduced cost,
    - improved quality,
    - lower risk.
- Wrong process?
    - Duplicated effort,
    - behind schedule.

Object-Oriented Programming (OOP) Paradigm

- Object Oriented programming (OOP) is a programming paradigm that uses the concept of **classes** and **objects**.
- Software programs are organized into simple, reusable pieces of code blueprints (classes), which are used to create individual instances of objects.
- Real-world objects are viewed as separate entities.
- Each has their own state and have built-in methods for maintaining the state.

Key Attributes of OOP

- Abstraction, Encapsulation, Inheritance, Polymorphism
- Ease of reuse
  - Speed implementation,
  - Facilitates maintenance,
  - Component-based approach
    - Easy to use existing components,
    - Easy to modify components to fit circumstances
- Natural way to view/model the world
  - Makes design quicker, easier & less error-prone

The world as we see it…

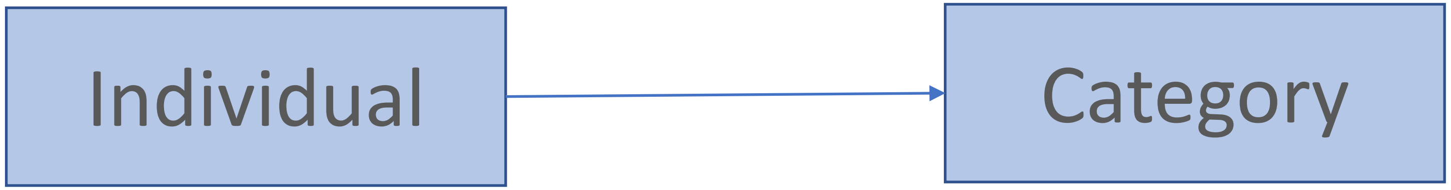
- Look around & what do you see?
    - **Things** (books, chairs, tables, people)

- Actually, you see **individual** things:
    - Your pencil, my textbook, that chair, that book, etc.

- The world is:
    - A set of things,
    - Interacting with each other

Describing the world

- Describe a particular person:
  - Ali has brown hair and green eyes, is 1.65 m tall and attended Bilkent University.
  - Eda has black hair and brown eyes, is 1.7 m tall and attended Hacettepe University.
- We can generalize the description for all people:
  - name, height, eye colour, university, …



Individual

Category

Writing Classes

- We've been using predefined classes from the Java API. Now we will learn to write our own classes.
- The class that contains the `main` method is just the starting point of a program.
- True object-oriented programming is based on defining classes that represent objects with well-defined characteristics and functionality.

Object-Oriented Programming

- An *object* represents an entity in the real world that can be distinctly identified.

- A student, a desk, a circle, a button, a loan can all be viewed as objects.

The State of an Object

- The *state* of an object (also known as its *properties or attributes*) is represented by *data fields* with their current values.

- For example:
  - An Employee object has a data field **salary**, which is the property that describes an employee.
  - A Rectangle object has data fields **width** and **height**, which are the properties that characterize a rectangle.

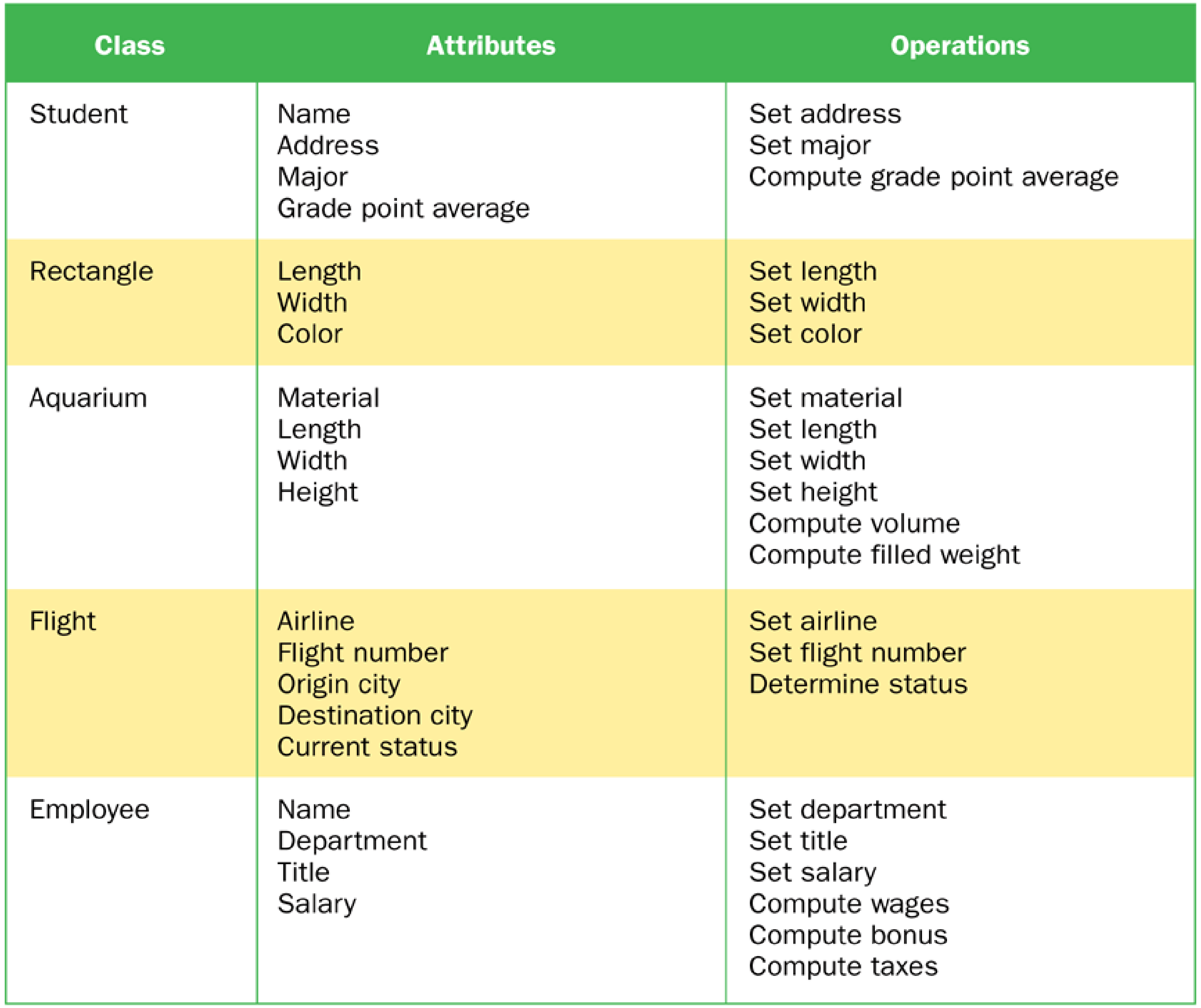
The Behavior of an Object

- The *behavior* of an object (also known as its *actions*) is defined by methods.
- To invoke a method on an object is to ask the object to perform an action.
- You may define a method named **getSalaryAfterTax()** for employee objects. The method is invoked on an employee object to get that employee's salary.

Classes and Objects

- Class:
  - A class describes a set of objects with the same
  - Objects of the same type are defined using a common class.
  - A class is a template, blueprint, or *contract* that defines what an object's data fields and methods will be.
- Object:
  - An object is an instance of a class.
  - You can create many instances of a class.
  - Creating an instance is referred to as *instantiation*.

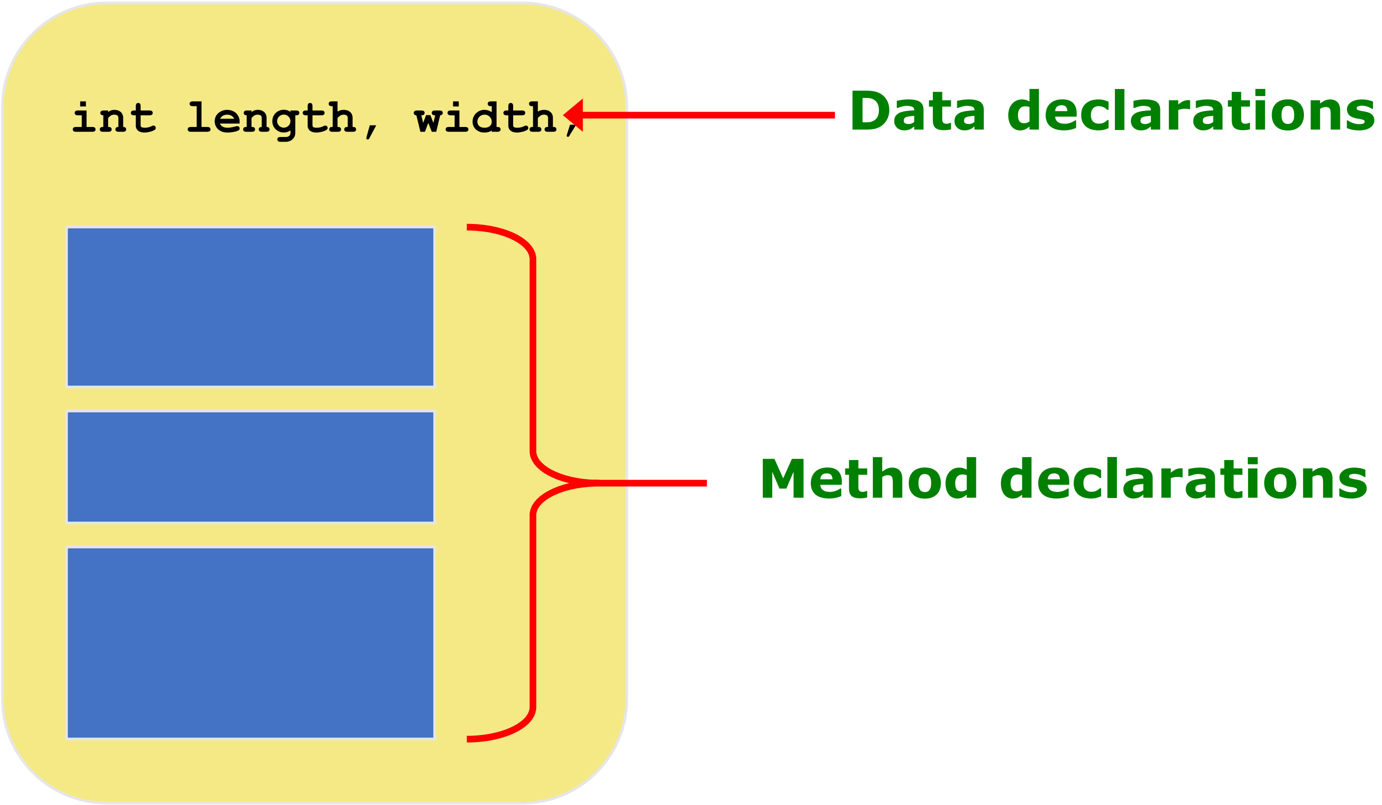
Examples of Classes



Class	Attributes	Operations
Student	Name Address Major Grade point average	Set address Set major Compute grade point average
Rectangle	Length Width Color	Set length Set width Set color
Aquarium	Material Length Width Height	Set material Set length Set width Set height Compute volume Compute filled weight
Flight	Airline Flight number Origin city Destination city Current status	Set airline Set flight number Determine status
Employee	Name Department Title Salary	Set department Set title Set salary Compute wages Compute bonus Compute taxes

Anatomy of a Class

- A class can contain data declarations and method declarations



Accessing Members of a Class

- **Within a class** you can access a member of the class the same way you would any other variable or method.
- **Outside the class**, a class member is accessed by using the syntax:
    - Referencing variables:

      `objectName.varName` example: **`arr.length`**
    - Calling non-static methods on objects:

      `objectName.methodName(params)`

      example: **`str.charAt(0);`**

Developing a Class: Rectangle1

- Refer to the files:
    - Rectangle1.java
    - Rectangle1Application.java

- Question: What happens if we change the length to a negative value? Is this logical?

- Question: What happens if we make the length and width private?

Visibility Modifiers

- In Java, we control access to an object's data and methods through the appropriate use of *visibility modifiers*

- A *modifier* is a Java reserved word that specifies particular characteristics of a method or data

- Java has three visibility modifiers: `public`, `protected`, and `private`

- The `protected` modifier involves inheritance, which we will not discuss in this course

Visibility Modifiers

- Members of a class that are declared with *public visibility* can be referenced anywhere.
- Members of a class that are declared with *private visibility* can be referenced only within that class.

Visibility Modifiers

- Visibility modifiers control access to the data members and allow for validation.

- Public variables are not recommended because they allow the client to modify the values directly, from outside the class.

- *Instance variables should not be declared with public visibility.*

- It is acceptable to give a constant public visibility, which allows it to be used outside of the class, because it cannot be changed.

Accessors and Mutators

- Because instance data is private, a class usually provides services to access and modify data values.

- An *accessor method* returns the current value of a variable.

- A *mutator method* changes the value of a variable.

- The names of accessor and mutator methods take the form `getX` and `setX`, respectively, where `X` is the name of the value.

- They are sometimes called “getters” and “setters”.

Improving a Class: Rectangle2

- Refer to the files:
  - Rectangle2.java
  - Rectangle2Application.java

- Question: What if we know the values of the data members when we instantiate the object?
- Question: What would happen if the object had 100 data members?

Constructors

- Constructors are special methods in a class.
- A *constructor* is used to set up an object when it is initially created (instantiated).
- The constructor creates the object and initializes its data members to default values.
- A constructor has the same name as the class.

Constructors

- A constructor is invoked with the new operator.
  - `Scanner scan = new Scanner(System.in)`
  - `Random randgen = new Random();`
- A constructor should initialize the instance variables.
- If the variables are not initialized, default values are used.
- A constructor does not have a return type.
- A constructor’s identifier (name) is the same as the class it constructs.

Constructors

- Note that a constructor has no return type specified in the method header, not even `void`

- A common error is to put a return type on a constructor.

- Each class has a *default constructor* that accepts no parameters.

Improving a Class: Rectangle3

- Refer to the files:
    - Rectangle3.java
    - Rectangle3Application.java

- Question: What happens if we pass negative values to the Constructor?

- Question: How can we improve the code to prevent this from happening?

- What happens if we output a Rectangle using the println method?

The toString Method

- It's good practice to define a `toString` method for a class
- The `toString` method returns a string that represents the object in some way
- It is called automatically when an object is concatenated to a string or when it is passed to the `println` method

Improving a Class: Rectangle4

- Refer to the files:
  - Rectangle4.java
  - Rectangle4Application.java

- Question: how do we calculate the area and perimeter of the rectangles? What happens if it is a complicated algorithm?

Service Methods

- Service methods are methods in the class that give an object its functionality.
- Often the service methods are the reason we create a class.
- Common operations with rectangles include calculating their width and height, therefore we include these methods in the class.

Improving a Class: Rectangle5

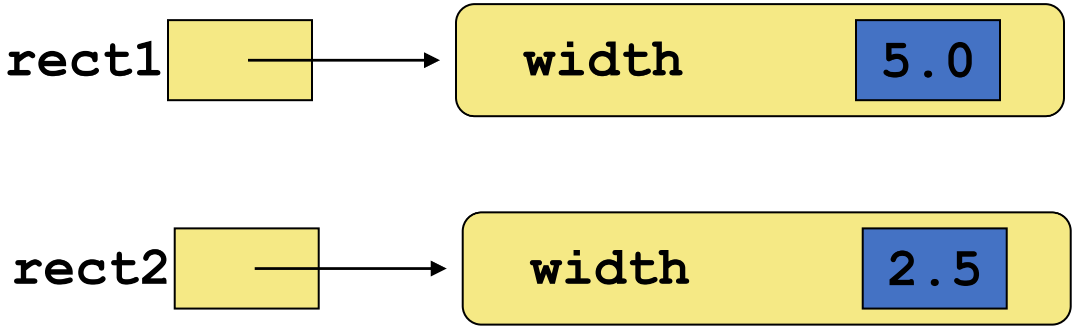
- Refer to the files:
  - Rectangle5.java
  - Rectangle5Application.java

State and Behavior

- Consider the rectangle
- Its state can be defined as its length and width.
- Its behaviors include calculation of its perimeter and area.
- We represent a rectangle by designing a class called `Rectangle` that models this state and behaviors.
- The class serves as the blueprint for a rectangle object.
- We can then instantiate as many rectangle objects as we need for any program.

Instance Data

- We can depict the two `Rectangle` objects from the `RectangleApplication` as follows:



Each object maintains its own `width` variable, and thus its own state

Instance Data

- A variable declared at the class level (such as `width`) is called *instance data.*
- Each instance (object) has its own instance variable.
- A class declares the type of the data, but it does not reserve memory space for it.
- Each time a `Rectangle` object is created, new `width and length` variables are created as well.
- The objects of a class share the method definitions, but each object has its own data space.
- That's the only way two objects can have different states.

Data Scope

- The *scope* of data is the area in a program in which that data can be referenced (used)
- Data declared at the class level can be referenced by all methods in that class.
- Data declared within a method can be used only in that method.
- Data declared within a method is called *local data.*
- In the `Rectangle` class, the variable `area` is declared inside the `getArea()` method -- it is local to that method and cannot be referenced anywhere else

Quick Check

What is the relationship between a class and an object?

Quick Check

What is the relationship between a class and an object?

A class is the definition/pattern/blueprint of an object. It defines the data that will be managed by an object but doesn't reserve memory space for it. Multiple objects can be created from a class, and each object has its own copy of the instance data.

Quick Check

Where is instance data declared?

What is the scope of instance data?

What is local data?

Quick Check

Where is instance data declared?

At the class level.

What is the scope of instance data?

It can be referenced in any method of the class.

What is local data?

Local data is declared within a method and is only accessible in that method.

Encapsulation

- We can take one of two views of an object:
  - ***internal*** - the details of the variables and methods of the class that defines it
  - ***external*** - the services that an object provides and how the object interacts with the rest of the system
- From the external view, an object is an *encapsulated* entity, providing a set of specific services.
- These services define the *interface* to the object.

Encapsulation

- One object (called the *client*) may use another object for the services it provides.
- The client of an object may request its services (call its methods), but it should not have to be aware of how those services are accomplished.
- Any changes to the object's state (its variables) should be made by that object's methods.
- We should make it difficult, if not impossible, for a client to access an object’s variables directly.
- That is, an object should be *self-governing*.

Encapsulation and Visibility

- Encapsulation is maintained using visibility modifiers.
- All variable data members in a class should be made private, with getters and setters to allow public access, where appropriate.
- Constant data members may be public, as they cannot be changed.

Methods and Visibility Modifiers

- Methods that provide the object's services are declared with public visibility so that they can be invoked by clients.

- Public methods are also called *service methods.*

- A method created simply to assist a service method is called a *support method.*

- Since a support method is not intended to be called by a client, it should not be declared with public visibility.

Visibility Modifiers

	public	private
Variables	Violate encapsulation	Enforce encapsulation
Methods	Provide services to clients	Support other methods in the class

Method Overloading

- Method Overloading: allows a class to have more than one method with the same name, if their argument lists are different.
- Association of method call to the method body is known as binding.
- **Static Binding** happens at compile time – the compiler determines which method to invoke at runtime.
- There are three ways to overload a method:
  - Number of parameters are different
  - Data type of parameters are different.
  - Sequence of parameter (data types) are different.
- What is ***not*** method overloading?
  - Different return type.

Method Overloading

- Known as static binding.
- Binding is the association of method call to the method body.
- **Static Binding** happens at compile time – the compiler determines which method to invoke at runtime.
- Why overloading?
  - Overloaded methods give flexibility - to call a similar **method** with different types of data.
  - Increase the readability and cleanliness of code.

- See `Student.java`

Constructor Overloading

- Constructors can be overloaded, like methods.
- Constructor overloading means having more than one constructor with different parameter lists.
- If a class does not define a constructor, a default constructor is inserted.
- The default constructor initializes the data members to default values.
- If a non-default constructor is defined, the default constructor (if necessary) should be defined as well using constructor overloading.

Support Methods

- Service methods are part of the public interface of a class.
- Objects are created to use their functionality (services), therefore service methods are public.
- Methods that are mean to support the functionality of a class, but that are not meant to be invoked from outside the class are sometimes called support methods.
- Support methods are part of the implementation of a class, not part of the interface.
- Support methods have `private` visibility.