Lecture 18

- Covers
  - Defining classes, attributes and methods
  - Constructors
  - Local variables
- Reading: Savitch 4.1

Lecture overview

- Concepts of object and class
- How the concepts are put to work
  - Define classes
  - Create objects
  - Use (manipulate) objects
- Summary (of class structure)
- Further example

- The concepts of object and class

Object-oriented concepts revisited

- Class
- Object
- Attribute
  - Instance attributes vs class attributes
- Operation / method
  - Instance methods vs class methods
- Instantiation
- Encapsulation/information hiding, interface
- Message passing

Will see how to use them in programming

Objects

- The concept of an object, in real life, is very general
- The only requirement for something to be an object is that it can be distinguished from other things
- Thus, virtually everything is an object

Objects

- Objects can be
  - Tangible things (e.g. tables, books, toasters)
  - People (e.g. employees, students)
  - Organisations (e.g. universities, companies, departments)
  - Non-tangible things (e.g. accounts, events)
  - etc.

Objects

- In Java, the generality of the real-life concept of object is more or less preserved
  - Things maintained in Java programs are either primitive data values or objects
  - We can define classes to represent real-life objects

Object - a definition

- In Java, as in the object-oriented paradigm in general

> an object is an entity that has a state (represented by its attributes or fields) and a set of behaviours (represented by its methods)

Classes

- Objects with common properties are often grouped and considered together
- This gives rise to the concept of a class
- A class is a blueprint, or a template, to describe and generate objects

Elements of a Java class

- The basic elements of a class are its
  - Attributes
  - Methods
  - Constructors
    - Constructors are special "methods" that are used when we create instances of the class

Abstractions

- Objects and classes (especially in programming) are abstractions
  - Simplified versions / representations of the real things
  - Only features relevant (to some purpose) are represented

- How the concepts are put to work

How the concepts are put to work

- The concepts are put to work in the following sequence of activities
  - Defining classes
  - Creating objects
  - Manipulating objects

Defining classes

- In real life, the concept of object may precede that of class
- The same thing may happen in the mind of a programmer
- But, in programming, we must always define the classes first

Defining classes

- To define a class is to define its
  - Attributes
  - Constructors
  - Methods

Creating objects

- Once a class is defined, we can create its objects (instances)
- We create instances of a class (instantiate the class) with the new operator which utilises a constructor of the class

Manipulating objects

- Once objects are created, we can manipulate them by sending messages to them
- These messages request the objects to perform certain methods
- An object can perform only methods "associated with it" (i.e. defined in its class or one of the super classes of the class)

Example

- Create and test a class representing a subject, that
  - Has a name, a lecturer, a quota and a current enrolment size
  - Allows users to enrol a student in the subject, un-enrol them from the subject, or display the details of the subject
- Rules
  - Cannot enrol a student when quota has been reached
  - Cannot un-enrol a student when there are no students in the class

Defining the class Subject

- To define the Subject class and test it, we go through the following typical steps
  - Sketch a model of the class
  - Define the class header
  - Define the attributes
  - Define the constructors
  - Define the methods

Sketching a model of the class

Subject	class name
String name String lecturer int quota int currentEnrolment	attributes
Subject (subjectName, lecturerName, maxQuota) void enrolStudent( ) void unEnrolStudent( ) void displaySubjectInfo( )	methods

Specifying the class header

```
public class Subject
{

}
```

Declaring the attributes

```
public class Subject
{
        private String name;
        private String lecturer;
        private int quota;
        private int currentEnrolment;
}
```

- Note that generally the attributes are declared to be private (more later)

Defining the constructors

```
public Subject(String subjectName,
                    String lecturerName, int maxQuota)
{
        name = subjectName;
        lecturer = lecturerName;
        quota = maxQuota;
        currentEnrolment = 0;
}
```

Constructor

- When a class is instantiated, a special initialisation method called a constructor is automatically invoked
- The constructor is defined as a method within the class definition
- It has the same name as the class itself
- In the example, it sets the initial values of the attributes to the values specified by the user creating the instance

```
Subject s = new Subject("Chemistry", "Dr Jekyll", 20);
```

Defining the methods

- To enrol a student
- To un-enrol a student
- To display subject information

Method to enrol a student

```
public void enrolStudent( )
{
    System.out.print("Enrolling student… ");
     if ( currentEnrolment < quota )
    {
        ++currentEnrolment;
        System.out.println("Student enrolled in " + name);
    }
    else
    {
        System.out.println("Quota reached, enrolment failed");
    }
}
```

method heading

method body

Method to un-enrol a student

```
public void unEnrolStudent( )
{
    System.out.print("Un-enrolling student… ");
    if ( currentEnrolment <= 0)
    {
        System.out.println("No students to un-enrol");
    }
    else
    {
        --currentEnrolment;
        System.out.println("Student un-enrolled from " + name);
    }
}
```

Method to display subject info

```
public void displaySubjectInfo( )
{
    System.out.println("Subject name: " + name);
    System.out.println("Lecturer: " + lecturer);
    System.out.println("Quota: " + quota);
    System.out.println("Currently enrolled: " +
    currentEnrolment);

    int availablePlaces = quota – currentEnrolment;
    System.out.println("Can accept " + availablePlaces +
                       " more students");
}
```

Local variables

- availablePlaces is defined in the displaySubjectInfo method and is only available to that method
- It is created when the displaySubjectInfo method is invoked, and destroyed when that method ends
- Its scope is the method displaySubjectInfo
- It is not an attribute of the Subject class or its instances, nor is it available for use in any other method defined by Subject

- Creating and manipulating objects

Testing the Subject class

- Define a launcher (driver) class to test Subject
  - This class has a main( ) method
  - In the main method, we create instances of Subject and send messages to them

Example

```
public class SubjectTester
{
    public static void main(String[ ] args)
    {
        // Test 1: Create a subject and display the subject
        Subject physics;
        physics = new Subject("Physics Principles", "Dr Who", 100);
        physics.displaySubjectInfo( );
    }
}
```

- *Observe the output*

Invoking a method

- To invoke a method in an object, we send it a message
- In Java, we specify the name of the object to receive the message and specify which of that object's methods is to be invoked by using the dot operator
- The receiver object is on the left hand side (lhs) of the dot, the method's name is on the right hand side (rhs) of the dot

```
physics.displaySubjectInfo( );
```

Example

```
// Test 2: Enrol a student and display the subject
physics.enrolStudent( );
physics.displaySubjectInfo( );
```

- *Observe the output*

Example

```
// Test 3 - Make an invalid un-enrol request and see how it
//          is handled (i.e. un-enrol the current one student
//          and then try to un-enrol another student)
physics.unEnrolStudent( );
physics.unEnrolStudent( );
physics.displaySubjectInfo( );
```

- *Observe the output*

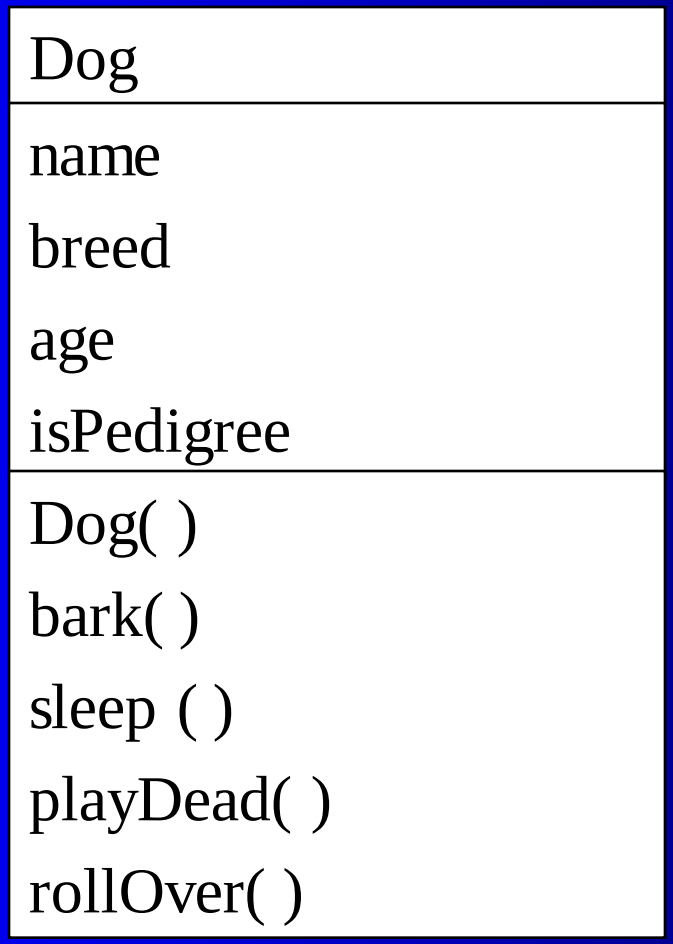
Example

```
// Test 4 – Test the upper boundary on the quota
for (int j = 0; j < 100; ++j)
{
   physics.enrolStudent( );
}
physics.displaySubjectInfo( );
physics.enrolStudent( );
physics.displaySubjectInfo( );
```

- *Observe the output*

- Summary

UML class summary



class name

attributes

operations

Java class summary

```
public class Dog
{
    private String name;
    private String breed;
    private int age;
    private boolean isPedigreeDog;

    public Dog( )
    {
        // initialisation code
    }
    public void sleep( )
    {
        // code for sleeping behaviour
    }
    public void playDead( )
    {
        // code for playing dead
    }
    public void rollOver( )
    {
        // code for rolling over
    }
}
```

Attributes that describe the state of a dog

Constructor to initialise Dog objects

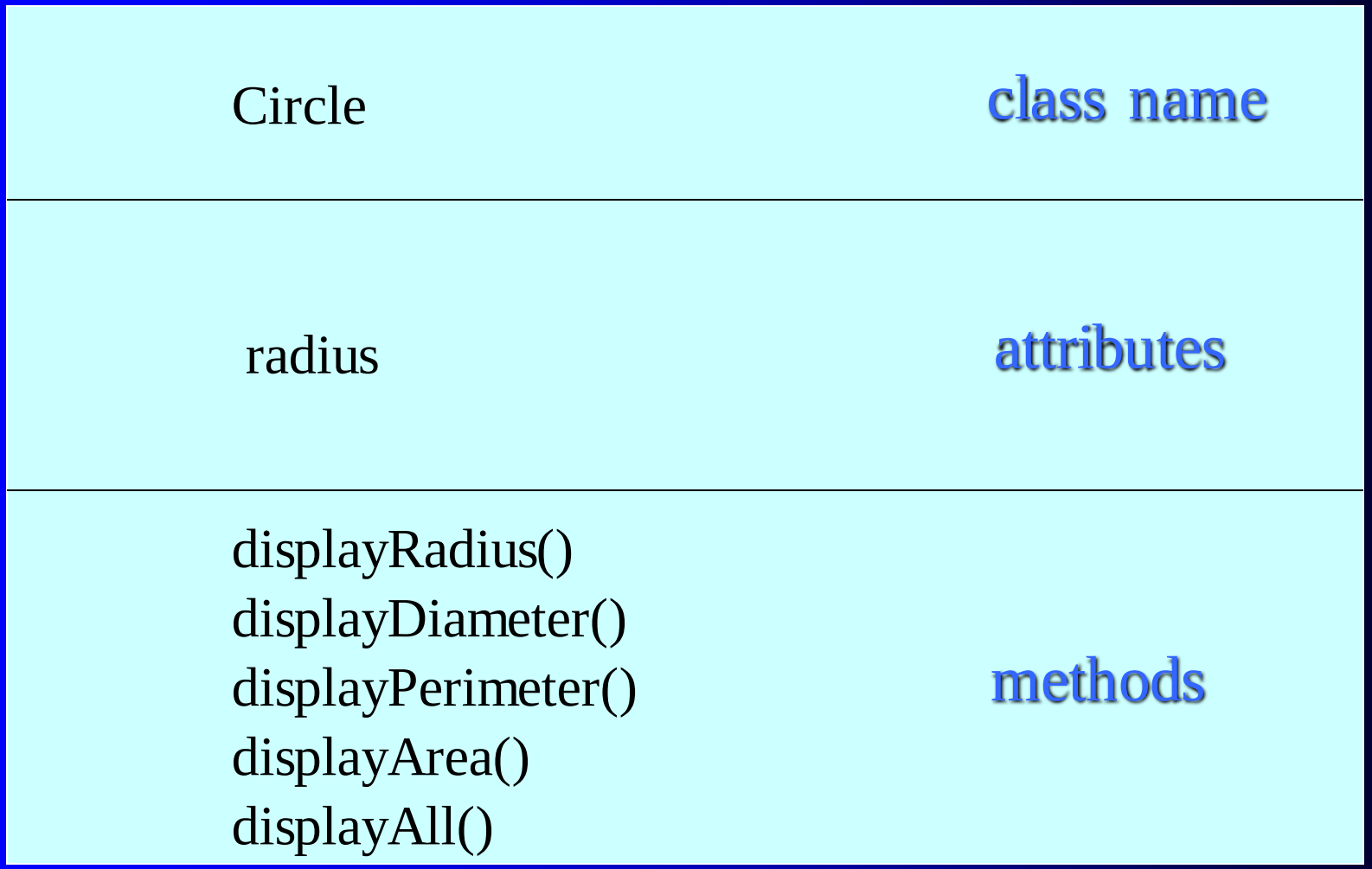
Methods that describe the behaviour of a dog

- Further example

Class exercise

- Define a class that describes a circle, which
  - Has a radius
  - Allows the user to
    - display the radius
    - calculate and display the diameter,
    - calculate and display the perimeter,
    - calculate and display the area,
    - and display all the measurements at once

Sketching a model for the class



Specify the class header

Declare the attributes

Define a constructor

Define the methods

- to display the radius
- to calculate and display the diameter
- to calculate and display the perimeter
- to calculate and display the area
- to display all details

Method to display the radius

Method to display the diameter

Method to display the perimeter

Method to display the area

Method to display all the details

Using methods from the same object

- When a method of an object wants to invoke a method in itself (the object), it must send a message to itself
- It does not need to specify the receiver object but can simply give the name of the method to be called
- displayPerimeter( )

Java files and separate compilation

- Each Java class needs to be in a separate file which has the same name as the class but with a .java extension
- A class can be compiled to byte code before a program exists to use the class
- When you create a launcher class that uses this class, you do not need to recompile the class
- If all the classes in your program are located in the same directory, the Java interpreter finds the right class when it is needed
- Later we will look at using classes in other directories

Next lecture

- Methods that return a value
- void methods
- Parameter passing (call-by-value)