

# Fundamentals of Programming

---

## Laboratory 9

### Object oriented programming II



# PP vs OOP (I)

## **Procedural(structured) Programming**

- separation of data and functionality
- variables are operated on by subroutines

## **Object Oriented Programming**

- integration of data and functionality
- objects serve as a combination of data (attributes, state) and functionality (behaviors)
- classes contain the design specifications for objects

# PP vs OOP (II)

## Procedural(structured) Programming

- + good for efficient creation of simple programs by a single programmer
- bad for large and complex programs made by many programmers
- difficult to maintain, modify and expand

## Object Oriented Programming

- + good for efficient creation of large, modular programs by many programmers
- + good for creating GUI interfaces
- + easy to maintain, modify and expand
- unnecessary for very small programs

# Main concepts in OOP

**A**bstraction

**P**olymorphism

**I**nheritance

**E**ncapsulation



# Abstraction

Generalizing complex things into a limited but informative description.

Classes can be specifically defined as **abstract**. Such classes are never meant to be used to create any objects, they are templates for other classes (inheritance).

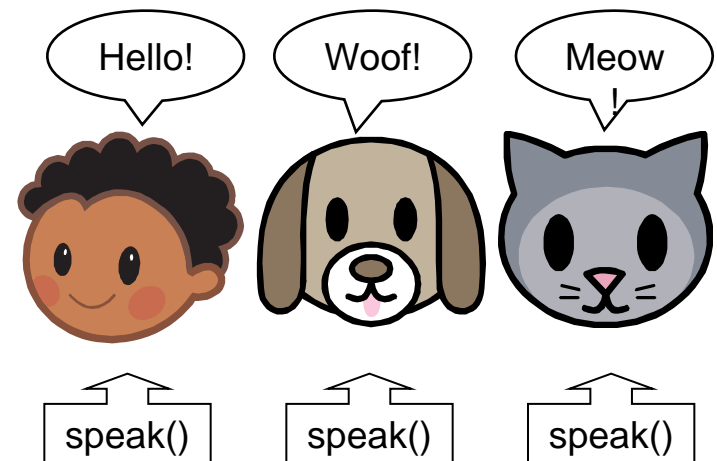


# Polymorphism

The same commands (method names) can perform different actions depending on what object they are called from and what parameters are passed.

Sub-classes can **override** inherited methods and create their own versions with different behavior.

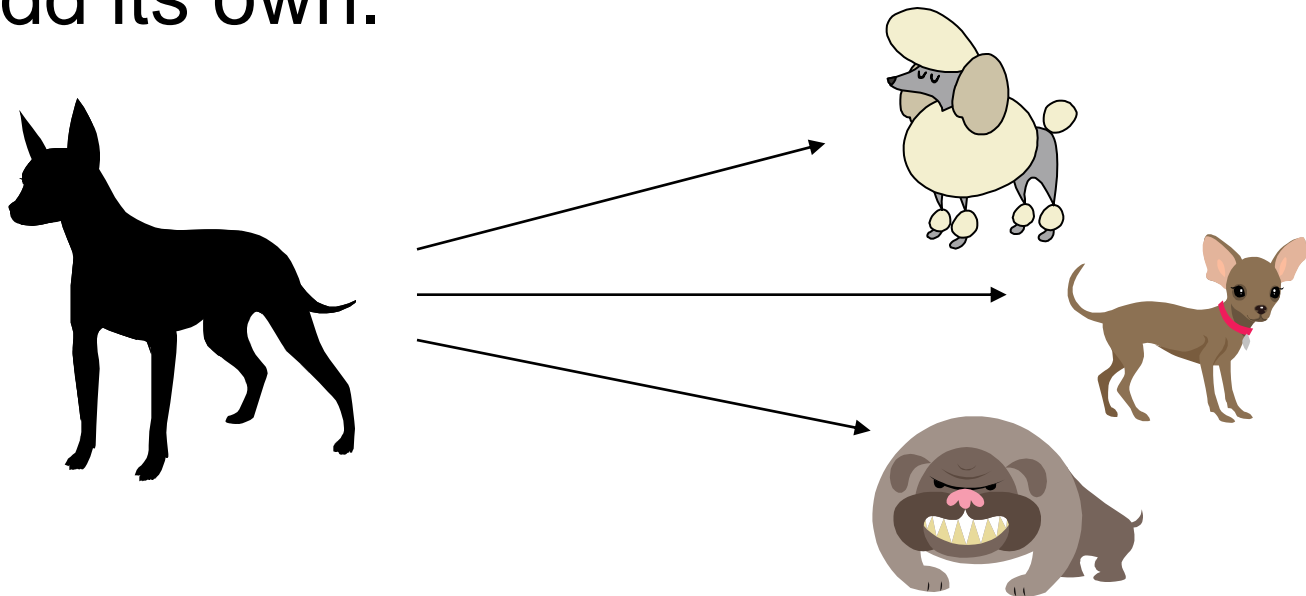
Methods can be **overloaded** by methods with the same name but different parameters.



# Inheritance

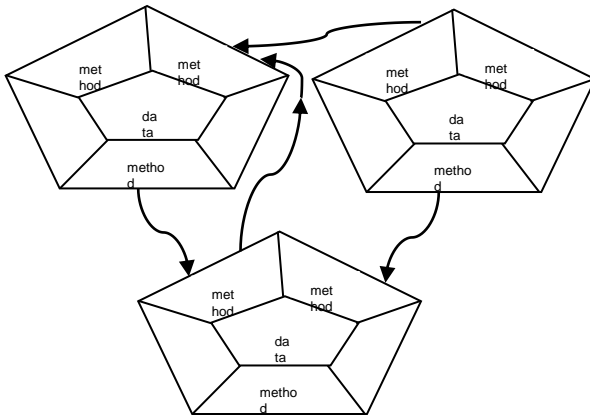
Creating more specialized sub-classes which expand/modify a parent class.

The inheriting class contains the same methods and variables as its parent class, but can modify them or add its own.



# Encapsulation

Hiding/protecting data (and sometime methods) inside a class and its objects. Clearly defining what attributes are **public** (accessible to all classes), **protected** (accessible only to classes inheriting from the given class) or **private** (accessible to the owner class only)







# Inheritance II

The keyword “extends” is used to describe that a class inherits from another.

```
class MountainBike extends Bicycle {  
    // new fields and methods defining a  
    // mountain bike would go here  
}
```

MountainBike is now called a **subclass/child** of Bicycle.  
Bicycle is the **superclass/parent** of MountainBike.

Objects of the MountainBike class will have all the attributes and behaviors defined for all Bicycle objects and can have additional ones declared in the class body.

# Polymorphism II

## Overriding

Subclasses can create their own versions of behaviors or attributes from the superclass. A subclass simply creates a method with the same name and as a method in the parent class and writes a new body for it.

## Overloading

A method in the same class or a subclass can be created with an identical identifier, but different parameters.

# Inheritance and overriding

```
class Rectangle
{
    int height, width;
    public void askForSides()
    {
        Scanner input = new Scanner(System.in) ;

        System.out.println("Input height:");
        height = input.nextInt();

        System.out.println("Input width:");
        width = input.nextInt();
    }
}
```



# Inheritance and overriding 2

```
class Square extends Rectangle
//subclass
{
    public void askForSides()
    //overridden method
    {
        Scanner input = new Scanner(System.in);
        System.out.println("Input side length:");
        height = input.nextInt();
        width = height;
    }
}
```

# Inheritance and overriding 3



Somewhere else in the program:

```
Rectangle obj1 = new Rectangle();
```

```
Square      obj2 = new Square();
```

```
obj1.askForSides();
```

```
obj2.askForSides();
```

# Abstract classes

Abstract classes cannot be instantiated, i.e. objects of such classes cannot be created.


A class defined as **abstract** is intended to be a superclass and only objects of its subclasses will be created.

Methods can also be abstract. They have no body, and are intended to be overridden in subclasses. If a class contains any abstract methods, it has to be made abstract as well.


If we want to create objects of a subclass of an abstract class, all abstract methods **must** be overridden.

# Abstract class example

```
abstract class Figure
{
    abstract public float area();
}
```



```
class Rectangle extends Figure
{
    int height, width;
    public float area()
    {
        return height*width;
    }
}
```



# Programming exercise 1

Write a program that calculates areas of geometrical figures. Your program should ask the user if he wants to calculate the area of a rectangle, triangle, or a circle.

After the user chooses one of the options (make the rectangle the default option if a wrong input is given) the program should create a single object of the appropriate class and perform all further actions using that object's behaviors (no matter what type of an object it is).





# Programming exercise 1

## (step by step)



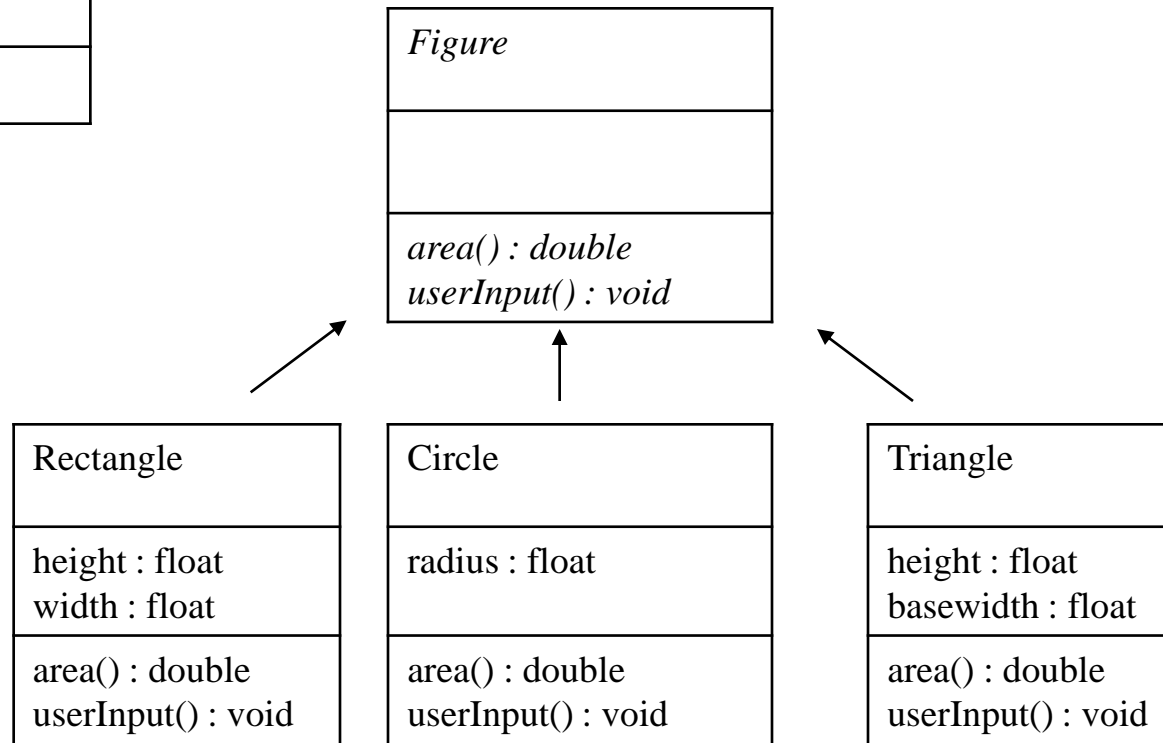
- 1) create a public starting class and a main method as usual, leave it empty for now
- 2) create an abstract class called **Figure** (use keyword **abstract**)
- 3) in the body of that class create the abstract methods **userInput()**, **area()** (area returns a double)
- 4) create three subclasses **Rectangle**, **Triangle**, **Circle** (use keyword **extends**)
- 5) in each of the subclasses create private variables to hold the appropriate dimensions (such as height, width or radius)
- 6) in each of the subclasses override the methods **userInput()**, **area()**
- 7) back in the main method create three objects (arectangle, acircle, atriangle)
- 8) ask the user to choose a figure (input an integer)
- 9) use an if/else or switch structure to call the appropriate object's user input method
- 10) print out the object's area



# Programming exercise 1 (diagram)



class name
attributes
behaviors



# Programming exercise 2

Create a starting program called GradeCalculator and inside it add a second class called Student:

Student

```
studentName : String
studentLastName : String
studentid, gradeGL1, gradeGL2,
gradeQ1, gradeQ2, gradeQ3,
bonus : int
```

```
Student(String,String)
calculatePercent() : double
checkRanges : boolean
checkPassing() : boolean
```



# Programming exercise 2

Ranges of values:

Grades for GLabs from 0 to 60, grades from quizzes from 0 to 20, laboratories+homeworks from 0 to 35

Conditions for passing:

A student must have received over 30 points in at least one G-Lab and a over 30 out of the 60 points in the three quizzes.

Grade calculation formula:

$$\text{Grade\_percent} = ((\text{gradeGL1} + \text{gradeGL2}) / 2 + \text{gradeQ1} + \text{gradeQ2} + \text{gradeQ3} + \text{gradeLabs}) / 155.0$$

In the main method provide a user interface for inputing the values for a student, with a looped question if he wants to modify one of the values. Display a warning if any values are in the wrong range.

