

## Introduction to Object-Oriented Programming (OOP)

Student Name

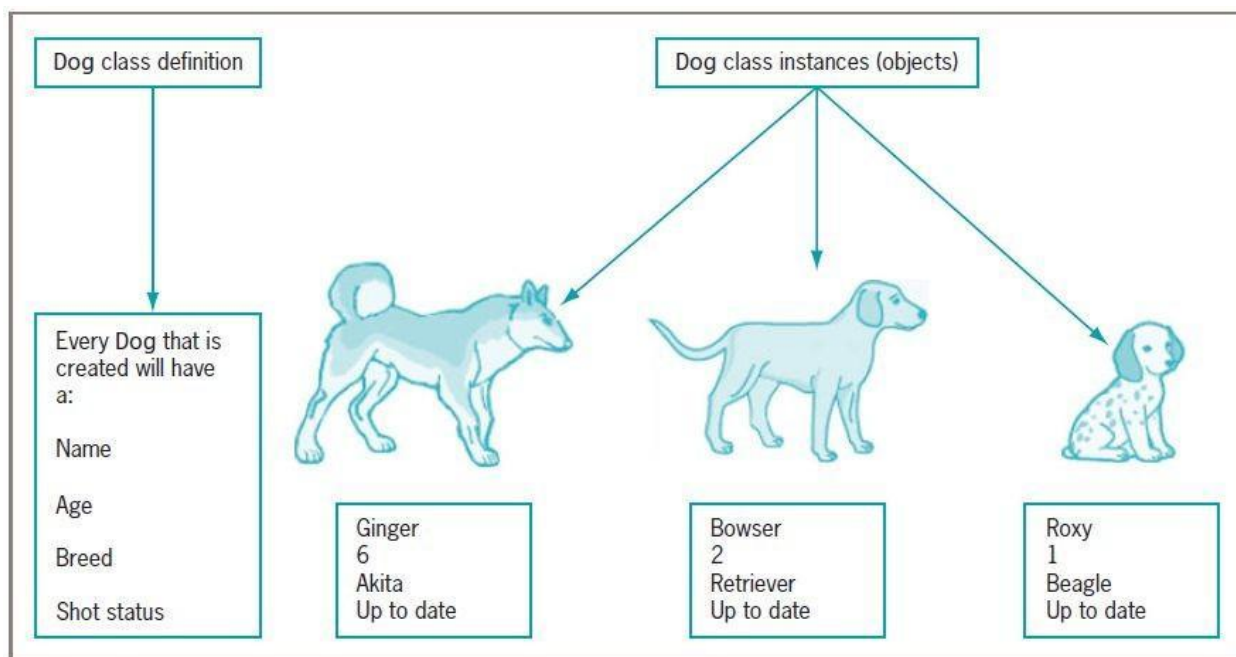
University

Class Name

Instructor name

## Introduction to Object-Oriented Programming (OOP)

Object-oriented programming (OOP) is a programming paradigm that organizes software design around data rather than functions and logic. An object is a data field with its own set of properties and behavior. Object-oriented programming (OOP) focuses on the objects that developers desire to handle rather than the logic that is required to handle them (Rai, 2019). The figure below shows OOP applied to dog class. This kind of programming is ideally suited to big, complicated, and frequently updated or maintained projects.



### Pillars of OOP

Encapsulation is a term that refers to the process of each object's implementation and state being kept private within a defined boundary or class. Other objects have no access to this class and no power to make modifications, but they can call a list of public functions or methods. This data-hiding feature improves software security and prevents inadvertent data damage.

## Code

### Main.java

```
public class Main {
    public static void main(String[] args) {
        Person myObj = new Person();
        myObj.setName("John");
        System.out.println(myObj.getName());
    }
}
```

### Person.java

```
public class Person {
    private String name;

    // Getter
    public String getName() {
        return name;
    }

    // Setter
    public void setName(String newName) {
        this.name = newName;
    }
}
```

## Output



John

Abstraction is a term used to describe something that does not expose its internal mechanisms (implementations), rather exposing an interface to call the hidden code. This technique allows developers to make modifications and additions more readily over time.

### Code

```
// Abstract class
abstract class Animal {
    // Abstract method (does not have a body)
    public abstract void animalSound();
    // Regular method
    public void sleep() {
        System.out.println("Zzz");
    }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
    public void animalSound() {
        // The body of animalSound() is provided here
        System.out.println("The pig says: wee wee");
    }
}

class Main {
    public static void main(String[] args) {
        Pig myPig = new Pig(); // Create a Pig object
        myPig.animalSound();
        myPig.sleep();
    }
}
```

### Output

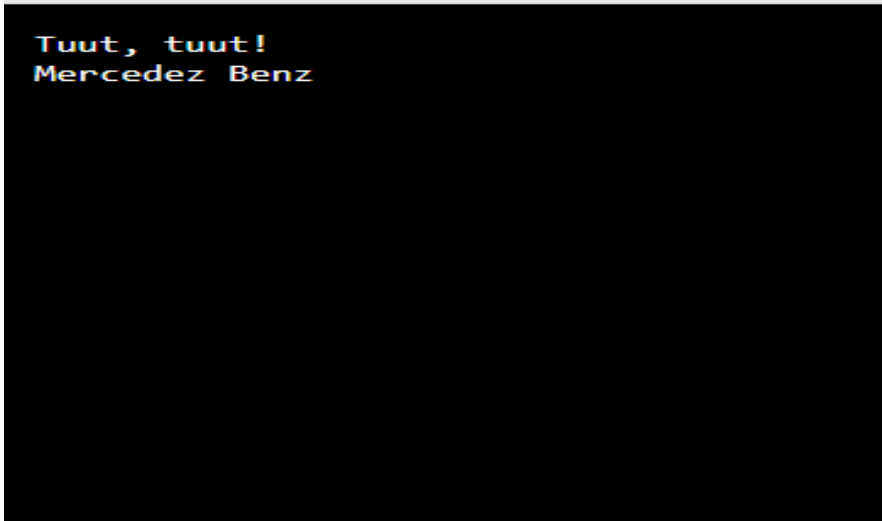
```
The pig says: wee wee  
Zzz
```

Inheritance is a term that refers to two items having a relation with each other. Developers can establish relationships and subclasses between items, allowing them to reuse similar logic while preserving a distinct hierarchy. This feature of OOP forces a more complete data examination, shortens development time and provides greater accuracy.

### Code

```
class Vehicle {  
    protected String brand = "Mercedez";  
    public void honk() {  
        System.out.println("Tuut, tuut!");  
    }  
}  
  
class Car extends Vehicle {  
    private String modelName = "Benz";  
    public static void main(String[] args) {  
        Car myFastCar = new Car();  
        myFastCar.honk();  
        System.out.println(myFastCar.brand + " " + myFastCar.modelName);  
    }  
}
```

## Output

A screenshot of a terminal window with a black background and white text. The text consists of two lines: "Tuut, tuut!" on the first line and "Mercedes Benz" on the second line.

Polymorphism is a term used to describe many forms. Depending on the situation, objects can take on multiple forms. The program will determine which meaning or usage is required for each object execution, reducing redundant code requirements.

## Code

```
class Animal {
    public void animalSound() {
        System.out.println("The animal makes a sound");
    }
}

class Pig extends Animal {
    public void animalSound() {
        System.out.println("The pig says: wee wee");
    }
}

class Dog extends Animal {
    public void animalSound() {
        System.out.println("The dog says: bow wow");
    }
}

class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myPig = new Pig();
        Animal myDog = new Dog();

        myAnimal.animalSound();
        myPig.animalSound();
        myDog.animalSound();
    }
}
```

## Output

```
The animal makes a sound  
The pig says: wee wee  
The dog says: bow wow
```

### **Advantages of Object-Oriented Programming**

OOP has a better software development productivity, since it helps in separating duty-based program development, and it is very extensible. In OOP, objects can be reused within programs, and based on these features, OOP provides a better software development productivity.

OOP provides improved maintainability software; this makes OOP an easier concept to manage and maintain. Also, OOP provides faster development capabilities. Due to the reuse functionality, OOP is a faster development platform. Other advantages of OOP include a considerably lower cost of development and the provision of higher software quality.

### **Disadvantages of Object-Oriented Programming**

OOP applications take larger amounts of programming, as they require more lines of code as compared to procedural applications. Moreover, OOP programs are slow compared to procedural programs. OOP is also not the best programming application for solving other problems that need functional, procedural logic programming.



## References

- Rai, L. (Ed.). (2019). Introduction to object-oriented programming and C++. In *Programming in C++: Object Oriented Features* (pp. 11–46). De Gruyter.  
<https://doi.org/10.1515/9783110593846-002>