

## Programming Fundamentals II

### Lab 9&10: Assignment

ID:

Name: **Abobaker Ahmed Khidir Hassan**

Department: **CS**

### Assignment Guidelines:

- Feel free to use any IDE for completing this laboratory assignment.
- This lab assignment comprises two exercises .
- Please ascertain that your code adheres to proper formatting and is adequately commented.
- Submit the code for each problem under its corresponding exercise number in a .java.(e.g Exercise1.zip)).
- **Make sure to fill in all the boxes in this document; this is mandatory!**  
Attach the completed document (Lab\_9\_10\_Assignment.docx or Lab\_9\_10\_Assignment.pdf) along with your code files (Exercise1.zip & Exercise2.zip); **this is mandatory!**

**Note:** *If you have any inquiries, please feel free to reach out to us via the discussion platform accessible to participants of this lab.*

## **FYI:**

- **Inheritance:** allows classes to inherit properties and methods from other classes.
- **Polymorphism:** allows methods to do different things based on the object it is acting upon.
- **Dynamic Binding:** refers to the JVM determining at runtime which method to invoke.
- **Object Casting:** allows you to convert one type of object reference to another.

## Exercise 1: UML diagrams, Inheritance (*The Triangle class*)

### Important Notes:

- ★ You can find this exercise in Chapter 11: Programming Exercises, page: 469, Exercise: 11.1 of the reference.
- ★ You can find the `GeometricObject.java` class code in Chapter 11 of the reference (Section: 11.2 Superclasses and Subclasses, page: 435), also you can find it in the codes folder of ([Java Code Files for Practical Laboratory 10](#) inside the [TestCircleRectangle.zip](#)) code files in the LMS.

(*The Triangle class*) Design a class named `Triangle` that extends `GeometricObject`. The class contains:

- Three `double` data fields named `side1`, `side2`, and `side3` with default values 1.0 to denote three sides of a triangle.
- A no-arg constructor that creates a default triangle.
- A constructor that creates a triangle with the specified `side1`, `side2`, and `side3`.
- The accessor methods for all three data fields.
- A method named `getArea()` that returns the area of this triangle.
- A method named `getPerimeter ()` that returns the perimeter of this triangle.
- A method named `toString()` that returns a string description for the triangle.

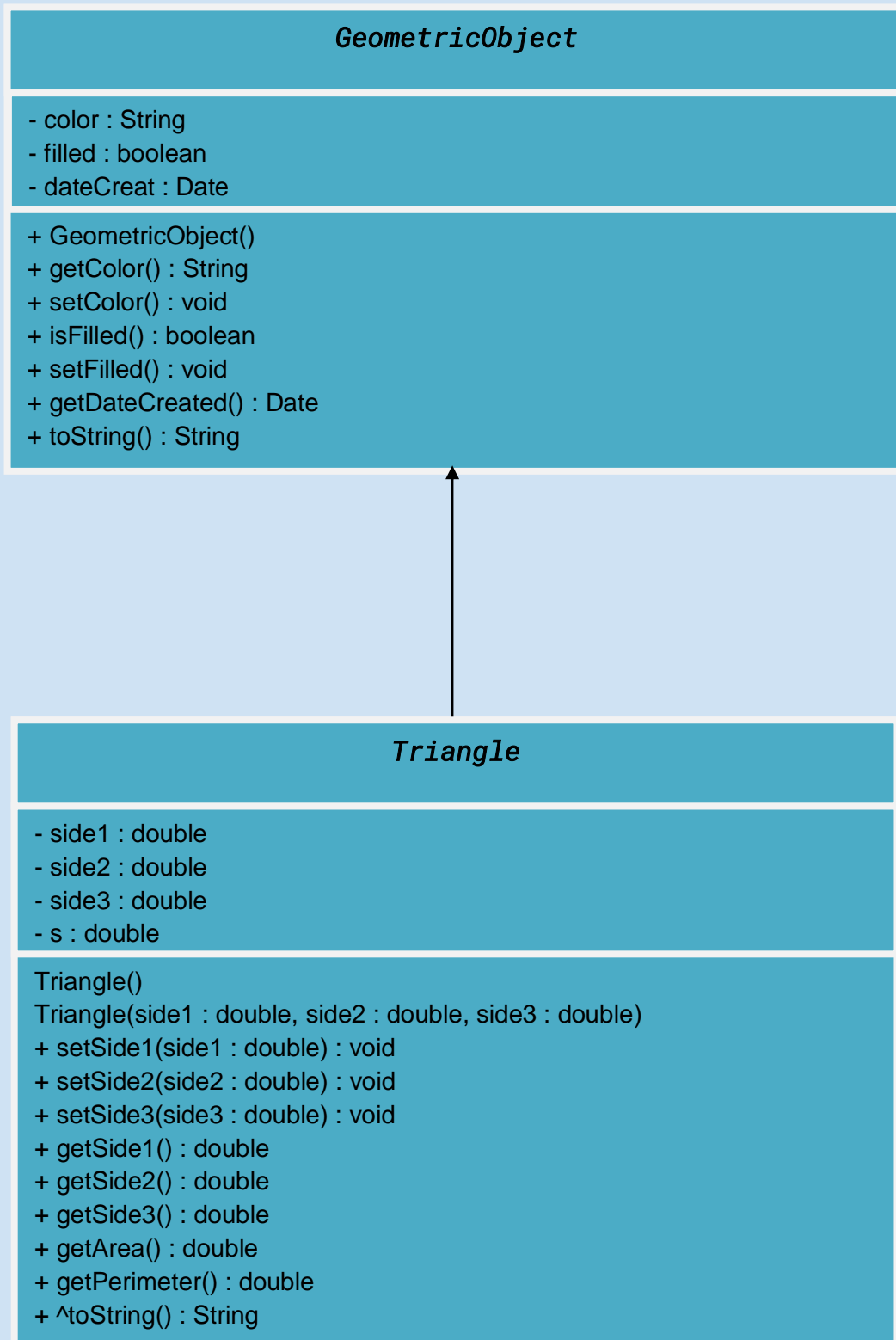
For the formula to compute the area of a triangle, see Programming Exercise 2.19.

The `toString()` method is implemented as follows:

```
return "Triangle: side1 = " + side1 + " side2 = " + side2 + " side3 = "
+ side3;
```

Draw the UML diagrams for the classes `Triangle` and `GeometricObject` and implement the classes. Write a test program that prompts the user to enter three sides of the triangle, a color, and a Boolean value to indicate whether the triangle is filled. The program should create a `Triangle` object with these sides and set the color and filled properties using the input. The program should display the area, perimeter, color, and true or false to indicate whether it is filled or not.

Insert your drawing of the UML diagrams for the classes *Triangle* and *GeometricObject* of your class design here:



*// Copy and paste your code of **GeometricObject.java** into this box:*

```
import java.util.Date;
public class GeometricObject {
    private String color = "white"; private boolean filled; private Date dateCreated;

    /** Construct a default geometric object */
    public GeometricObject() { dateCreated = new java.util.Date(); }

    /** Construct a geometric object with the specified color * and filled value */
    public GeometricObject(String color, boolean filled) {
        dateCreated = new java.util.Date(); this.color = color; this.filled = filled; }

    /** Return color */
    public String getColor() { return color; }

    /** Set a new color */
    public void setColor(String color) { this.color = color; }

    /** Return filled. Since filled is boolean, * its getter method is named isFilled */
    public boolean isFilled() { return filled; }

    /** Set a new filled */
    public void setFilled(boolean filled) { this.filled = filled; }

    /** Get dateCreated */
    public java.util.Date getDateCreated() { return dateCreated; }

    /** Return a string representation of this object */
    public String toString() {
        return "created on " + dateCreated + "\n color: " + color + " and filled: " + filled; }
}
```

*// Copy and paste your code of **Triangle.java** into this box:*

*// 1- Design a class named Triangle that extends GeometricObject.*

```
class Triangle extends GeometricObject{
```

```
    //■ Three double data fields with default values 1.0.
```

```
    private double side1 = 1;
```

```
    private double side2 = 1;
```

```
    private double side3 = 1;
```

```
    //■ A no-arg constructor that creates a default triangle.
```

```
    Triangle(){}
```

```
    //■ A constructor with specified three sides.
```

```
    Triangle(double side1, double side2, double side3){
```

```
        this.side1 = side1;
```

```
        this.side2 = side2;
```

```
        this.side3 = side3;
```

```
    }//constructor
```

```
    //■ The accessor methods for all three data fields.
```

```
        //Setters:
```

```
        public void setSide1(double side1){ this.side1 = side1; }//setSide1
```

```
        public void setSide2(double side2){ this.side2 = side2; }//setSide2
```

```
        public void setSide3(double side3){ this.side3 = side3; }//setSide3
```

```
        //Getters:
```

```
        public double getSide1(){ return this.side1; }//getSide1
```

```
        public double getSide2(){ return this.side2; }//getSide2
```

```
        public double getSide3(){ return this.side3; }//getSide3
```

```
    //■ A method named getArea() that returns the area of this triangle.
```

```
    public double getArea(){
```

```
        double s = (side1 + side2 + side3)/2;
```

```
        return Math.pow((s * (s - side1) * (s - side2) * (s - side3)),0.5);
```

```
    }//getArea
```

```
    //■ A method named getPerimeter() that returns the perimeter of this triangle.
```

```
    public double getPerimeter(){
```

```
        return ( side1 + side2 + side3 );
```

```
    }//getPerimeter
```

```
    //■ A method named toString() that returns a string description for the triangle.
```

```
    @Override
```

```
    public String toString(){
```

```
        return "Triangle: side1 = " + side1 + " side2 = " + side2 + " side3 = " + side3;
```

```
    }//toString
```

```
}//Triangle
```

```
/*  
Copy and paste the code of your Exercise program Test Class that contain the main  
method Exercise1.java into this box:  
*/
```

```
import java.util.*;  
public class Exercise1{  
  
    public static void main(String [] args){  
        Scanner input = new Scanner(System.in);  
  
        // 1- Prompt the user to enter the triangle  
        System.out.println("Enter 3 sides of your triangle please");  
        //Side 1  
        System.out.print("Side 1: "); double side1 = input.nextDouble();  
        //Side 2  
        System.out.print("Side 2: "); double side2 = input.nextDouble();  
        //Side 3  
        System.out.print("Side 3: "); double side3 = input.nextDouble();  
        //Color  
        System.out.print("Color: "); String color = input.next();  
        //Is filled?  
        System.out.print("Is it filled? (If yes enter T, else enter F): ");  
        String temp = input.next(); Boolean isFilled = false;  
        if(temp.equals("T") || temp.equals("t") || temp.equals("true")) isFilled = true;  
  
        System.out.println();  
  
        // 2- Create a Triangle object with these properties  
        Triangle triangle1 = new Triangle( side1, side2, side3);    //Set sides.  
        triangle1.setColor(color);    //Set the color.  
        triangle1.setFilled(isFilled);    //Set is it filled or not.  
  
        // 3- The program should display:  
        //The area:  
        System.out.println("The area: " + triangle1.getArea() + " unite square.");  
        //The perimeter:  
        System.out.println("The perimeter: " + triangle1.getPerimeter() + " unit.");  
        //color + is it filled or not:  
        System.out.println("The color: " + triangle1.getColor() + ".\nIs it filled? " +  
triangle1.isFilled() + ".");  
  
    }//main  
  
}// Exercise1
```

## Exercise 2: Inheritance, Polymorphism, Dynamic Binding, and Object Casting

In this lab exercise, you will explore the concepts of inheritance, polymorphism, dynamic binding, and object casting using Java. Follow the steps below to create the classes and complete the tasks. Run your code after each step to check your output and understand the concepts.

### Part 1: Inheritance:

#### Step 1: Create a Base Class

1. Create a class called `Animal`.
2. Add a string field `name`.
3. Add a constructor that initializes the `name` field.
4. Add a method `makeSound()` that prints "Animal makes a sound".

*// Copy and paste the code segment from step 1 into this box:*

```
// 1. Create a class called Animal.
class Animal{

    // 2. Add a string field name.
    private String name;

    // 3. Add a constructor that initializes the name field.
    Animal(){
        this.name = "Animal";
    }//constructor 1
    Animal(String name){
        this.name = name;
    }//constructor 2

    public void setName(String name){ this.name = name; }//setName
    public String getName(){ return this.name; }//getName

    // 4. Add a method makeSound() that prints "Animal makes a sound".
    public void makeSound(){
        System.out.println("Animal makes a sound");
    }//makeSound

}// Animal
```



## Step 2: Create Derived Classes

1. Create a class called **Dog** that extends **Animal**.
2. Override the **makeSound()** method to print "Dog barks".
3. Create a class called **Cat** that extends **Animal**.
4. Override the **makeSound()** method to print "Cat meows".

```
// 1. Create a class called Dog that extends Animal.
class Dog extends Animal{

    Dog(){ super("Dog"); }//constructor

    // 2. Override the makeSound() method to print "Dog barks".
    @Override
    public void makeSound(){
        System.out.println("Dog barks");
    }//makeSound
}//Dog

// 3. Create a class called Cat that extends Animal.
class Cat extends Animal{

    Cat(){ super("Cat"); }//constructor

    // 4. Override the makeSound() method to print "Cat meows".
    @Override
    public void makeSound(){
        System.out.println("Cat meows");
    }//makeSound

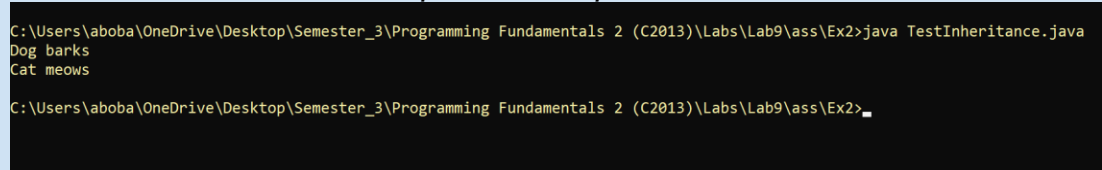
}//Cat
```

### Step 3: Test Inheritance

1. Create a `TestInheritance` class with a `main` method.
2. Create instances of `Dog` and `Cat`.
3. Call the `makeSound()` method on each instance and check the output.

```
/*  
Copy and paste the code segment from step 3 (Just code of step 3) into this  
box:  
*/  
  
// 1. Create a TestInheritance class with a main method.  
public class TestInheritance{  
  
    public static void main(String [] args){  
  
        // 2. Create instances of Dog and Cat.  
        Dog dog1 = new Dog();  
        Cat cat1 = new Cat();  
  
        // 3. Call the makeSound() method on each instance.  
        dog1.makeSound();  
        cat1.makeSound();  
    }  
}  
//main  
  
// TestInheritance
```

*Insert a screenshot of the output from Step 3 here:*



```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>java TestInheritance.java  
Dog barks  
Cat meows  
  
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>_
```

Dog barks  
Cat meows

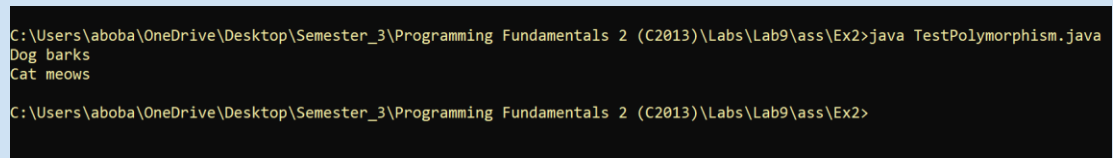
## Part 2: Polymorphism

### Step 4: Use Polymorphism

1. Create a `TestPolymorphism` class with a `main` method.
2. Declare variables of type `Animal` and assign them instances of `Dog` and `Cat`.
3. Call the `makeSound()` method on each variable and check the output.

```
/*  
    Copy and paste the code segment from step 4 (Just code of step 4) into this  
    box:  
*/  
  
// 1.    Create a TestPolymorphism class with a main method.  
public class TestPolymorphism{  
  
    public static void main(String [] args){  
        // 2.    Declare variables of type Animal and assign them instances of  
        Dog and Cat.  
        Animal dog = new Dog();  
        Animal cat = new Cat();  
  
        // 3.    Call the makeSound() method on each variable.  
        dog.makeSound();  
        cat.makeSound();  
    }  
}  
//main  
  
} //TestPolymorphism
```

*Insert a screenshot of the output from Step 4 here:*



```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>java TestPolymorphism.java  
Dog barks  
Cat meows  
  
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>
```

Dog barks  
Cat meows

## Part 3: Dynamic Binding

### Step 5: Understand Dynamic Binding

1. Create a `TestDynamicBinding` class with a `main` method.
2. Declare a variable of type `Animal`.
3. Assign it an instance of `Dog` and call `makeSound()`.
4. Assign it an instance of `Cat` and call `makeSound()`. Check the output.

```
/*
Copy and paste the code segment from step 5 (Just code of step 5) into this
box:
*/

// 1. Create a TestDynamicBinding class with a main method.
public class TestDynamicBinding{

    public static void main(String [] args){

        // 2. Declare a variable of type Animal.
        Animal animal;

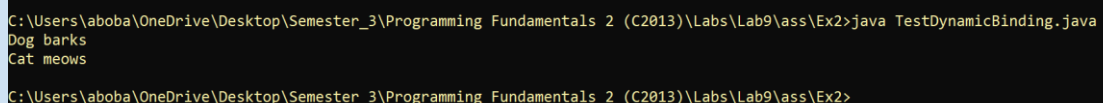
        // 3. Assign it an instance of Dog and call makeSound().
        animal = new Dog();
        animal.makeSound();

        // 4. Assign it an instance of Cat and call makeSound().
        animal = new Cat();
        animal.makeSound();

    } //main

} //TestDynamicBinding
```

*Insert a screenshot of the output from Step 5 here:*



```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>java TestDynamicBinding.java
Dog barks
Cat meows
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>
```

Dog barks  
Cat meows

## Part 4: Object Casting:

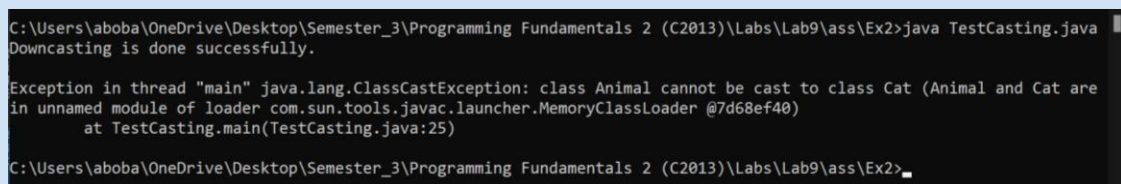
### Step 6: Practice Object Casting

1. Create a `TestCasting` class with a `main` method.
2. Perform upcasting and downcasting between `Animal`, `Dog`, and `Cat`.
3. Check the output and handle `ClassCastException` (Think of **Exception Handling** from Lecture 12 “Exception Handling” and lab 12).

**Note:** If you need further explanation on the casting process, please refer to Lab 10, Part 3.

```
/*  
    Copy and paste the code segment from step 6 (Just code of step 6) into this box:  
*/  
// 1. Create a TestCasting class with a main method.  
public class TestCasting{  
    public static void main(String [] args){  
  
        Animal animal1 = new Animal();    Dog dog1 = new Dog();  
        Animal animal2 = new Animal();    Cat cat1 = new Cat();  
  
        //2. Perform downcasting:  
        animal1 = dog1;  
        System.out.println("Downcasting is done successfully.\n"); //will be printed  
  
        //2. Perform upcasting:  
        cat1 = (Cat) animal2; //It looks like trying to store an integer in a byte.  
        System.out.println("Upcasting is done successfully.\n"); //Will not be printed  
    }  
}  
//main  
//TestCasting
```

*Insert a screenshot of the output from Step 6 here:*



```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>java TestCasting.java  
Downcasting is done successfully.  
  
Exception in thread "main" java.lang.ClassCastException: class Animal cannot be cast to class Cat (Animal and Cat are  
in unnamed module of loader com.sun.tools.javac.launcher.MemoryClassLoader @7d68ef40)  
    at TestCasting.main(TestCasting.java:25)  
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>
```

Downcasting is done successfully.

Exception in thread "main" java.lang.ClassCastException: class Animal cannot be cast to class Cat (Animal and Cat are in unnamed module of loader com.sun.tools.javac.launcher.MemoryClassLoader @7d68ef40) at TestCasting.main(TestCasting.java:25)

## Follow-Up Questions and Tasks:

1. What is the output when `animal.makeSound()` is called in `TestDynamicBinding`?

*Insert a screenshot of the output here:*

```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>java TestDynamicBinding.java
Dog barks
Cat meows

C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Ex2>
```

Dog barks  
Cat meows

2. Why does a `ClassCastException` occur in the `TestCasting` class?

*Write your answer here:*

Because `ClassCastException` is called by default when an casting error happened. Exactly here, we cannot cast an `Animal` object to a `Cat` object after the creation statement ( `Animal animal2 = new Animal()` ), because a `Cat` object is always an instance of `Animal`, but an `Animal` is not necessarily an instance of `Cat`.

## Tasks:

1. Extend the `Animal` class to include another derived class called `Bird` and override the `makeSound()` method.

```
/*
Copy and paste the code segment from this task into this box:
*/

// 1.a) Extend the `Animal` class to include another derived class called `Bird`.
class Bird extends Animal{
    Bird(){ super("Bird"); }//constructor

    // 1.b) Override the `makeSound()` method.
    public void makeSound(){
        System.out.println("Birds Sound");
    }//makeSound
}//Bird
```

2. Implement a scenario where upcasting and downcasting are used correctly with the `Bird` class.

There is two scenarios here of upcasting and downcasting:  
Without Errors - With an Excepted Error

#### 1- Without Errors:

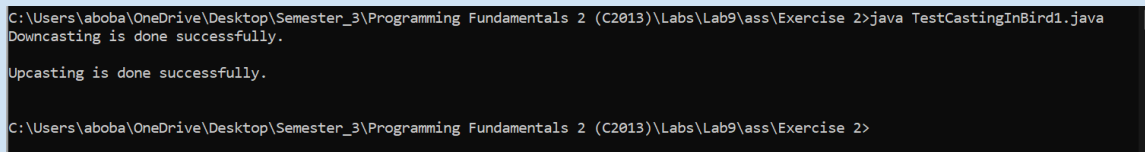
```
/*
Copy and paste the code segment from this task into this box:
*/
/*
It can be done correctly if an Animal object was storing a Bird object when we try to
assign another (or the same) Bird object to it.
*/
public class TestCastingInBird1{

    public static void main(String [] args){
        Animal animal3 = new Animal();    Bird bird1 = new Bird();

        try{
            //Downcasting:
            animal3 = bird1;
            System.out.println("Downcasting is done successfully.\n");

            //Upcasting:
            bird1 = (Bird) animal3;
            System.out.println("Upcasting is done successfully.\n");
        }//try
        catch(ClassCastException ex){
            System.out.println("Error: There is some error in the type casting!");
        }//catch
    }//main
}//TestCastingInBird1
```

*Insert a screenshot of the output here:*



```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Exercise 2>java TestCastingInBird1.java
Downcasting is done successfully.
Upcasting is done successfully.
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Exercise 2>
```

Downcasting is done successfully.

Upcasting is done successfully.

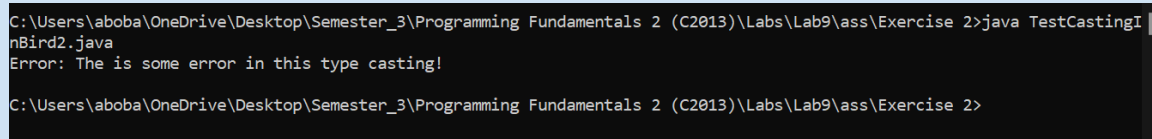
## 2- With an Excepted Error:

```
/*
Copy and paste the code segment from this task into this box:
*/
/*
    I changed the upcasting above the downcasting to print the statement in the
    excepted error.
*/
public class TestCastingInBird2{
    public static void main(String [] args){
        Animal animal3 = new Animal();    Bird bird1 = new Bird();
        try{
            //Upcasting:
            bird1 = (Bird) animal3; //It looks like tring to store an integer in a byte.
            System.out.println("Upcasting is done successfully.\n");

            //Downcasting:
            animal3 = bird1;
            System.out.println("Downcasting is done successfully.\n");

        }//try
        catch(ClassCastException ex){
            System.out.println("Error: The is some error in the type casting!");
        }//catch
    }//main
}//TestCastingInBird2
```

*Insert a screenshot of the output here:*



```
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Exercise 2>java TestCastingI
nBird2.java
Error: The is some error in this type casting!
C:\Users\aboba\OneDrive\Desktop\Semester_3\Programming Fundamentals 2 (C2013)\Labs\Lab9\ass\Exercise 2>
```

Error: There is some error in this type casting!

You should implement these steps, run your code, and observe the outputs to solidify your understanding of these concepts.

*Good Luck!*

**End of Lab!**