Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Objectives of today's lab:

Through this lab, students will examine how casting works in Java and learn about Abstract Class and Interface in Java with examples.

Create a Java project named `week10`

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Create a package named `basicoop3` in the project `week10`

Exercise 1: Upcasting and Downcasting

Upcasting: casting a subtype to a supertype, upward to the inheritance tree.

Downcasting: casting from base class to child class.

Task 1.1: Write the following program, save, compile, and run it.

/* file name should be `TestInheritance.java` in the `basicoop3` package*/

```
class Animal{
public void eat(){
System.out.println("Animal Eating!");
}
}
class Cat extends Animal{
public void meow (){
System.out.println("Meowing!");
}

public void eat(){
System.out.println("Cat Eating!");
}

}

class Dog extends Animal{
public void bark (){
System.out.println("Barking!");
}

public void eat(){
System.out.println("Dog Eating!");
}

}

public class TestInheritance{
public static void main(String args[]){
Animal animal = new Cat();
animal.eat();
```

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

```
Dog dog = new Dog();
animal = dog;
animal.eat();
}
}
```

If you can successfully run your program, you should see the following output:

```
Cat Eating!

Dog Eating!
```

Observation: First a Cat instance is created. Second, the Cat instance is assigned to a variable of type Animal. Now the Animal variable (reference) points to the Cat instance. This is possible because the Cat class inherits from the Animal class. Same things happen for the Dog class.

As you can see, it is possible to use an instance of some subclass as if it were an instance of its base class. This way, we don't need to know exactly what subclass the object is an instance of. We could treat e.g., both Cat and Dog instances as Animal instances.

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Task 1.2: Comment out the `eat()` method in the `Animal` class. Then save, compile, and run the code.

```
class Animal{
/*
public void eat(){
System.out.println("Animal Eating!");
}  */
}
```

Does your code work correctly? If not, why?

Now Un comment the `eat()` method in the Animal class. Then, invoke the `meow()` method before creating the Dog object.

```
 animal.eat();
 animal.meow();
 Dog dog = new Dog();
```

Does your code work correctly? If not, why?

These are common mistakes:

After casting, you will have access only to the current reference type class members even your object is from type Cat and your reference of type Animal. Any unique class member in class Cat will not be visible to you.

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Task 1.3: Now create the following object in the last working code.

```
Cat cat = new Animal();
```

Does your code work correctly? If not, why?

Here, an Animal instance is created. Second, the Animal instance is assigned to a variable of type Cat.

By definition Cat is-a Animal, but can you say Animal is-a Cat?

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Task 1.4: The downcasting is the casting from base class to child class and is not automatic. Write the following code in the `main()` method, save, compile, and run the program.

```
Animal animal = new Cat();

animal.eat();

/*down casting manually */

Cat cat = (Cat) animal;

cat.eat();
```

If you can successfully run your program, you should see the following output:

```
Cat Eating!

Cat Eating!
```

Now you remove `(Cat)` from the following

```
Cat cat = (Cat) animal;
```

So the statement becomes

```
Cat cat = animal;
```

Does your code work correctly? If not, why?

……

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Exercise 2: Abstract class

Basic concepts:

- **Abstract classes may or may not contain *abstract methods***
- **If a class contains at least one abstract method, then the class must be declared abstract**
- **If a class is declared abstract, it cannot be instantiated**
- **To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.**
- **If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.**
- **If you don't implement an abstract method (declared in the abstract class) in the sub-class, then the sub-class becomes abstract too.**

Task 2.1: Write the following code, save, compile, and run it.

/* file name should be `TestAbstract.java` in the `basicoop3` package*/

```
abstract class Animal{
public void eat(){
System.out.println("Animal Eating!");
}
}
class Cat extends Animal{
public void meow (){
System.out.println("Meowing!");
}

public void eat(){
System.out.println("Cat Eating!");
}

}

class Dog extends Animal{
public void bark (){
System.out.println("Barking!");
}

public void eat(){
System.out.println("Dog Eating!");
}

}

public class TestAbstract{
public static void main(String args[]){
Cat cat = new Cat();
cat.eat();
Dog dog = new Dog();
dog.eat();
}
}
```

If you can successfully run your program, you should see the following output:

```
Cat Eating!

Dog Eating!
```

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Task 2.2: Create the following object.

```
Animal animal = new Animal();
```

Does your code work correctly? If not, why? (If you don't know, read again the basic concepts given above)

Task 2.3: Modify the `eat()` method in the `Animal` class as below (make it abstract). Save, compile, and run the program.

```
public abstract void eat();
```

If you can successfully run your program, you should see the following output:

```
Cat Eating!

Dog Eating!
```

Task 2.4: Comment out the `eat()` method in the `Cat` class as below. Save, compile, and run the program.

```
class Cat extends Animal{
public void meow (){
System.out.println("Meowing!");
}

/*
public void eat(){
System.out.println("Cat Eating!");
} */

}
```

Does your code work correctly? If not, why? (If you don't know, read again the basic concepts given above)

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Task 2.5: In the following program, insert the missing code to get the expected output (at the end).

```
/* file name should be Student.java  in the basicoop3 package*/

abstract class Person {

	private String name;
	private String gender;

	public Person(String name, String gender){
		//code missing
	}

	//abstract method
	public abstract void study();

	@Override
	public String toString(){
		//code missing
	}

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

	   public String getName(String name)  {
		return this.name;
	}
}
```

```
public class Student extends Person {

	private int studentId;

	public Student(String name, String gender, int studentId)
{
		//code missing
	}

	@Override
	public void study() {
		if(studentId == 0){
				System.out.println("The person is Not a
Student");
		}else{
			System.out.println("The person is a Student");
		}
	}

	public static void main(String args[]){
		Person per1 = new Student("Tomas Maul","Male",0);
		Person per2 = new Student("Sheema","Female",13579);
		per1.study();
		per2.study();
		per2.setName("Sheema Das");
		System.out.println(per2.toString());
	}

}
```

Expected output:

```
The person is Not a Student
The person is a Student
Person's name is: Sheema Das and Gender is:
Female
```

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Exercise 3: Java Interface

Basic concepts:

- **A Java Interface is just a definition of behaviour, similar to a Java abstract class**
- **It is not a class but it is defined in a way similar to the class definition**
- **An Interface describes the functions and behaviors of a specific object type but doesn't implement them**
- **When a class implements an interface it has to implement all methods declared in that interface.**

Task 3.1: Write the following code, save, compile, and run it.

/* file name should be `TestInterface.java` in the `basicoop3` package*/

```
interface Anima{
  public abstract void eat();
}
class Cat implements Animal{
public void meow (){
System.out.println("Meowing!");
}


@Override
public void eat(){
System.out.println("Cat Eating!");
}
}

class Dog implements Animal{
public void bark (){
System.out.println("Barking!");
}

@Override
public void eat(){
System.out.println("Dog Eating!");
}

}

public class TestInterface{
public static void main(String args[]){
Cat cat = new Cat();
cat.eat();
Dog dog = new Dog();
dog.eat();
}
}
```

If you can successfully run your program, you should see the following output:

```
Cat Eating!
Dog Eating!
```

Week 10(19) OOSD1 Java lab exercises: Casting, Abstract class and Java Interface

Task 3.2: Create a default constructor in the `Animal` interface

```
interface Anima{
  public Animal(){};
  public abstract void eat();
}
```

Does your code work correctly? If not, why? (If you don't know, read again the basic concepts given above)

Task 3.3: Now modify the `eat()` method of the `Animal` Interface

```
interface Animal{
public void eat(){
System.out.println("Animal Eating!");
}
```

Does your code work correctly? If not, why? (Same reason as above?)

Congratulations! Today, you have learned other important features of the Java language. Now please help your fellow classmates who have been struggling to complete these exercises or you can work on your group coursework.