Introduction to Programming

Lab 10

Alaa Aldin Hajjar, Mahmoud Naderi, Marko Pezer,
Mosab Mohamed, Rawan Ali

Agenda

- Abstract classes
- Final classes
- Interfaces

Exercise 1 - Abstract classes

- Create the abstract class *Creature* with abstract methods *bear()* and *die()* and String field *name* equal to *null* and boolean *isAlive* equal to *false*. Also, create non-abstract method *shoutName()*, which should print the name, if it's not equal to null. Otherwise, it should print error message
- Create classes *Human*, *Dog* and *Alien* which should inherit the *Creature*. Override all abstract methods for all 3 classes differently
- For *bear()* method each of them should assign the *name* and print the message "The *[class name] [name]* was born"
- For *die()* method each of them should print the message "The *[class name] [name]* has died"
- Add a method *bark()* to a class *Dog*

Exercise 1- Abstract classes (cont.)

- Create the *AbstractClassDemonstration* class, to demonstrate the functionality
- Modify Exercise 1 *AbstractClassDemonstration* class, so that array of creatures of different types (*Human, Dog, Alien*) is created. For each element of the array call methods *bear()* and *die().*

Hint: you can use ArrayList instead of array

Discussion

1. Creature dog = new Dog();
2. dog.bark();

Exercise 2 - Final classes

Extend the previous exercise solution to include the following:

- Create class *Animal* which should inherit the *Creature*.
- Make classes *Human and Dog* inherit the *Animal* instead of *Creature.*
- Modify classes *Human and Dog* to prohibit them from being inherited further.

Exercise 3 - Interfaces

- Create an interface *Swimmable* with methods *swim()* and *stopSwimming()*
- Create an interface *Flyable* with methods *fly()* and *stopFlying()*
- Create an interface *Living* with default method *live()* that prints "*[class name]* lives"
- Create class *Submarine* which implements *Swimmable* and override methods
- Create class *Duck* which implements *Swimmable, Flyable* and *Living,* and override non-default methods
- Create class *Penguin* which implements *Swimmable* and *Living,* and override non-default methods
- Create the *InterfaceDemonstration* class, to demonstrate the functionality.

Hint: to stop swimming/flying creature has to be swimming/flying

Exercise 3 - Interfaces (cont.)

- Modify *InterfaceDemonstration* class, so that array of living objects of different types (*Duck, Penguin*) is created. For each element of the array call method *live().*

Discussion

- What should happen if *swim()* is called for the elements of this array?
- Can instance of a *Submarine* be added to this array?

Discussion

What is the difference between interfaces and abstract classes?

Discussion

Abstract class	Interface
1) Abstract class can **have abstract and non-abstract** methods.	Interface can have **only abstract** methods. Since Java 8, it can have **default and static methods** also.
2) Abstract class **doesn't support multiple inheritance**.	Interface **supports multiple inheritance**.
3) Abstract class **can have final, non-final, static and non-static variables**.	Interface has **only static and final variables**.
4) Abstract class **can provide the implementation of interface**.	Interface **can't provide the implementation of abstract class**.
5) The **abstract keyword** is used to declare abstract class.	The **interface keyword** is used to declare interface.

Discussion

Abstract class	Interface
6) An **abstract class** can extend another Java class and implement multiple Java interfaces.	An **interface** can extend another Java interface only.
7) An **abstract class** can be extended using keyword "extends".	An **interface** can be implemented using keyword "implements".
8) A Java **abstract class** can have class members like private, protected, etc.	Members of a Java interface are public by default.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

References

- Overview of Inheritance, Interfaces and Abstract Classes in Java | by Isaac Jumba | Medium
- Polymorphism in Java
- Problems