EU8-JAVA-OOP Concepts Abstraction

Abstraction

▼ What is Abstraction?

- Abstraction means we ***focus on the essentials rather than small details of the method*** *(hiding the implementations/details of the method).*
- *Only focus on what it is (name of the action/behavior), instead of how it's done.*

▼ It concentrates on name of the action/behavior, without thinking about implementation

- ***Focus on the essential***
- Ignore the irrelevant
- Ignore the unimportant

- Abstraction helps with ***"organizing the code", "reusing the code", "less duplicate code",***
- In Java, abstraction is achieved by ***abstract classes*** and ***interfaces***. We can achieve 100% abstraction using interfaces.

▼ Creating Abstract Classes

- ***abstract*** keyword is used to create abstract classes

- Goal is ***to provide reusable variables and methods to sub classes***

▼ An ***abstract class cannot be instantiated***

we cannot declare object using these classes, it is not concrete

public abstract class Browser{}
Browser browser = new Browser(); //Error

▼ ***Abstract class can have both abstract and instance method***

Abstract class doesn't have to contain only abstract methods.

▼ ***If there is an abstract method in a class, that class must be also abstract***

▼ ***an abstract class can be extended to another abstract class***

public abstract class A{}

public abstract class B extends A{}

▼ *an abstract class can be extended to another non-abstract class.*

public class A{}

public abstract class B extends A{}

▼ *Can a class extends multiple abstract classes?*

- *No, it can not*

public abstract class A{}

public abstract class B{}

public class C extends A,B{} //Error

Can I do this with Interface? YES

▼ *Can we add constructor in abstract class?*

Yes, we can.

▼ *If we cannot instantiate abstract class, how we can call the constructor?*

The constructor of abstract class can be called from a subclass using ***super***

▼ *Can we add static method into abstract class?*

- *Yes, we can*

public abstract class A {

public static void methodX (){

} }

▼ Creating Abstract Methods

- ***abstract*** keyword is used to create abstract method
- Abstract method ***does not have body/implementation, only have signature***
- ***An abstract method is meant to be overridden (can not be final, private, static)***
- Can be ***only placed in abstract class or interface***

▼ *Is it mandatory for abstract class to have abstract method?*

No, it is not. Abstract class can have 0 abstract method.

▼ Can abstract method be protected, private, and default?

- private: no
- default: yes
- protected: yes

▼ *Can a method be abstract and final in abstract class?*

- *No, it can not*

public abstract class Browser {

public abstract final void navigate(String url); //Error

public abstract void displayWebpage(); }

▼ *Can a method be abstract and static in abstract class?*

No, it can not , Only instance method can be overridden

▼ Creating Concrete Class

- ***A sub class of abstract class is called concrete class (opposite of abstract class)***
- ***A first concrete class must implement (= override) all inherited abstract methods***

▼ *Where do you add implementation?*

- *Is added in first concrete class*
- ***While adding implementation, all overriding rules should be followed****.*

▼ *When an abstract class extends another abstract class, does it require to override/implement abstract methods from parent?*

- *No, it does not require*
- *First concrete sub class is required to implement all abstract methods*

▼ Extending Another Abstract Class

- An abstract class can extend another abstract class. If so it is ***optional to implement abstract methods from abstract super class***.
- A first concrete class must implement all inherited abstract methods.

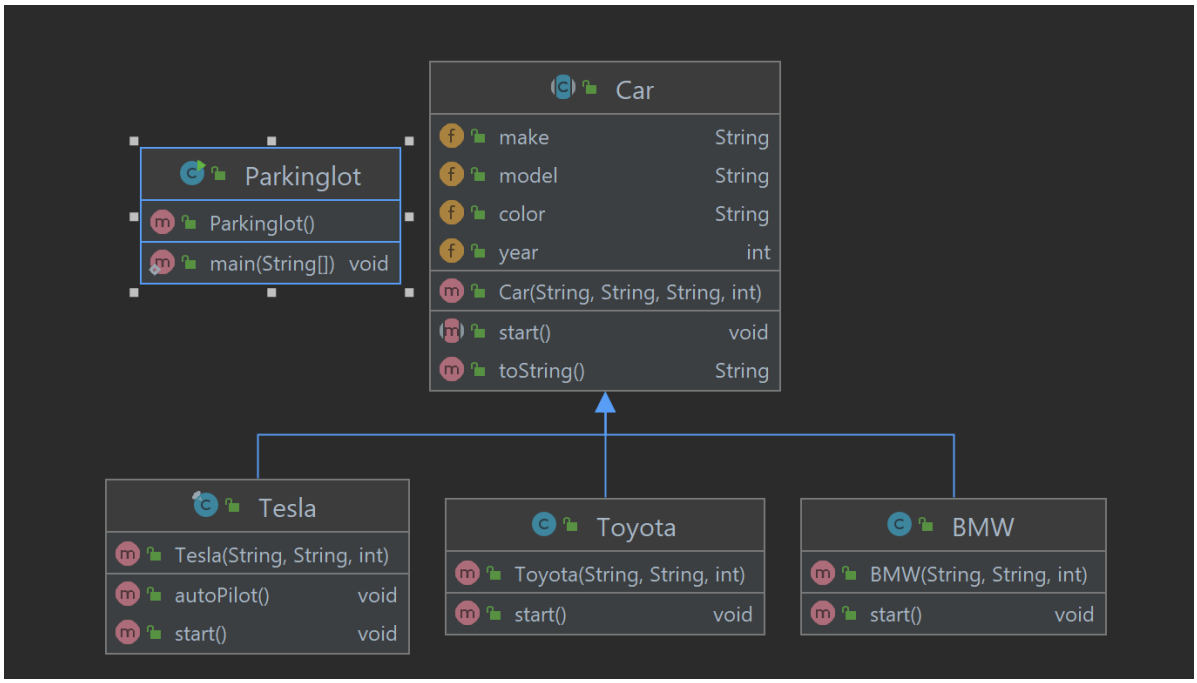
▼ Abstract Class Rules Review

- Abstract classes cannot be instantiated directly
- Abstract classes may be defined with any number, including zero, of abstract and non-abstract methods
- Abstract classes may not be marked as private or final
- An abstract class that extends another abstract class inherits all of its abstract methods as its own abstract methods
- The first concrete class that extends an abstract class must provide an implementation for all of the inherited abstract methods

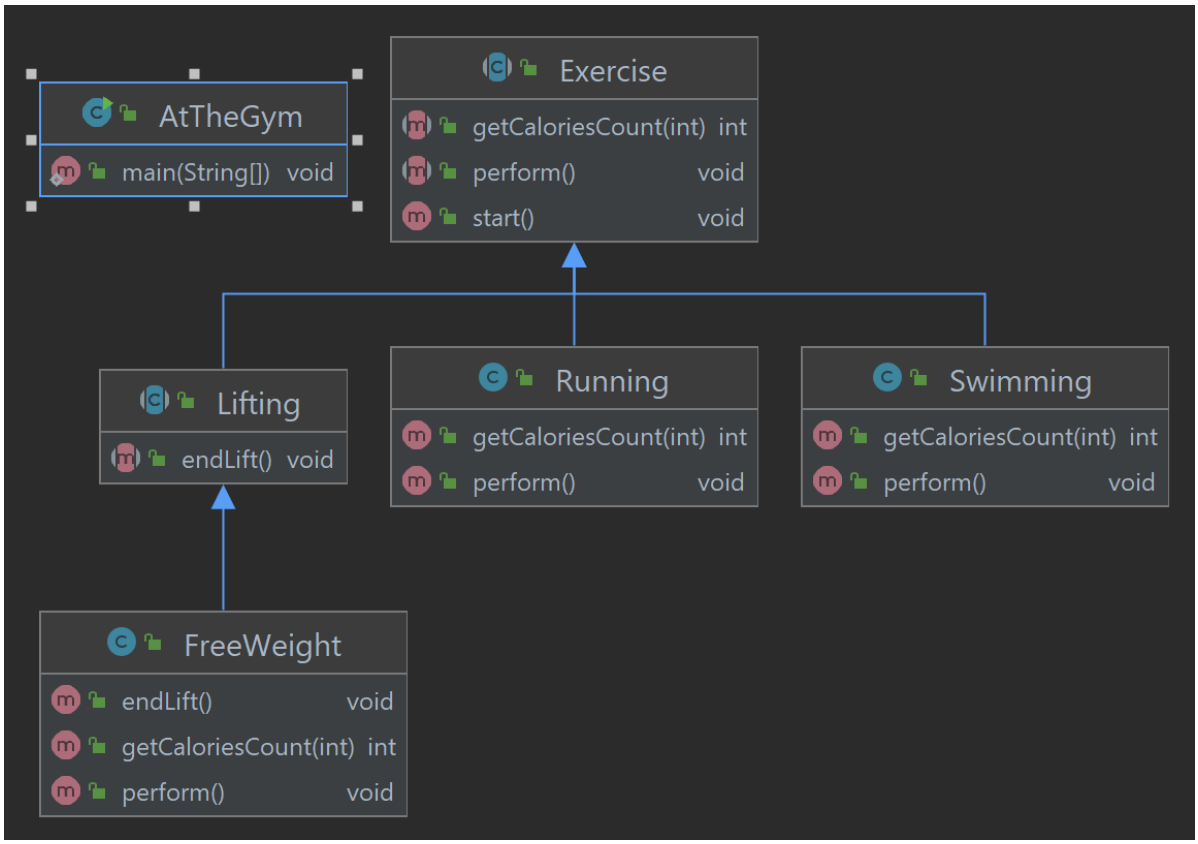
▼ Abstract Method Rules Review

- Abstract methods may only be defined in abstract classes or interface
- Abstract methods can not be declared private or final
- Abstract methods must not provide a method body/implementation in the abstract class for which it is declared
- Implementing an abstract method in a subclass follows the same rules for overriding a method
- ***Variables cannot be abstract***

▼ Example 1



▼ Example 2



Interfaces

▼ What is interface?

In many ways an interface is similar to an abstract class, but its intent is to specify common
behavior for objects of **related classes or unrelated classes**.

Not a class but a blueprint of a class, specifies the additional behaviors/features that the class needs to implement.

Inheritance forms “IS A” relationship between classes

Implementing Interfaces forms “IS A KIND OF” relationship, less strong relationship

▼ ***Contract between a class and outside world***

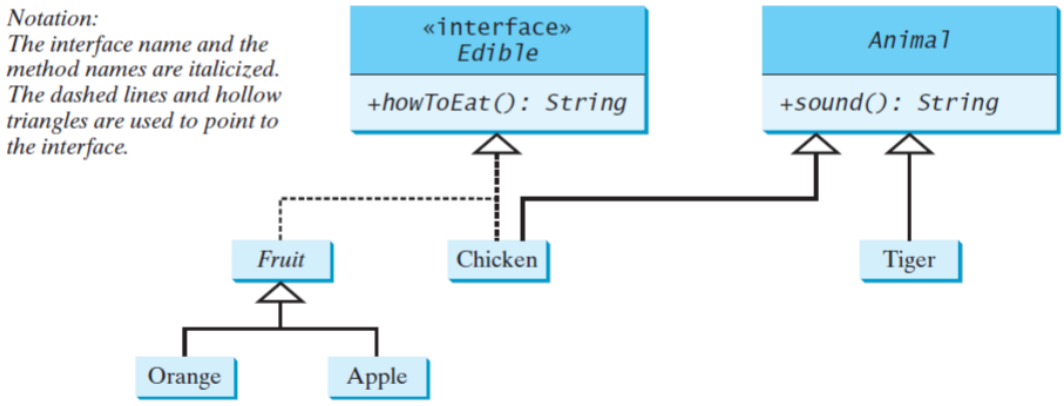
Implementing an interface allows a class to become more formal about the behavior it promises to provide. Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler. If your class implements an interface, all methods defined by that

interface must appear in its source code before the class will successfully compile.

▼ ***Meant to be implemented.***

- Provide *set of abstract methods (before java 8)*
- Most interfaces have a group of related empty methods
- The class provides the behaviors included in the interface

▼ Interface Example



▼ ***Can have: (what?)***

- ***Access modifiers: public***
- ***Constant Variables: final, static***
- ***Type of Methods: abstract, static, default***
- S***tatic and default methods: sub-classes don't have to implement default and static methods.***

▼ ***Can not have: (what?)***

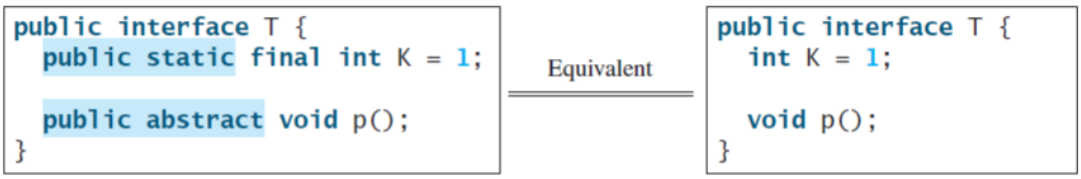
- ***Access modifiers: private, default, protected***
- ***Instance variables & methods***
- ***Final methods***
- ***Constructor***

- ***Blocks (static or instance)***

▼ ***Implicitly have:***

- ***Fields (variables) : public, static, final***
- ***All methods : public***
- ***Except static and default methods : abstract***

▼ Since all data fields are public static final and all methods are public abstract in an interface, Java allows these modifiers to be omitted. Therefore the following interface definitions are equivalent:



▼ Creating an Interface

▼ An interface looks similar to a class, except the keyword ***interface*** is used instead of the keyword class

```
public interface Electric {

    public abstract void charge();

    void test(); // public abstract -- by default


}
```

▼ Implementing Multiple Interfaces

- Class can ***extend*** only one superclass, but java allows a class to implement multiple interfaces.
- When a class implements multiple interfaces, it must ***provide the methods specified by all of them (all methods have to be overridden.***

```
public class MyClass implements Interfacse1,Interfacse2,Interfacse3{}
```

▼ Fields in Interface

▼ An interface can contain field declarations, but all fields in an interface are ***treated as public, static and final***.

```
public interface Electric {

    public static final boolean HAS_BATTERIES = true;

    boolean HAS_BATTERIES2 = false; // public final static


    public abstract void charge();

    void test(); // public abstract -- by default


}
```

▼ Implementing Interface

▼ When you want a class to implement an interface, you use the ***implements***

```java
public interface Displayable{

    public abstract void display();

}

public class Person implements Displayable{

    public void display(){

        //code
    }
}
```

- A class can ***extends another class and implements interface(s) same time***.

▼ If a class both extend a class and implement an interface, extends should come first then implements keyword.

```java
public class Student extends Person implements Teachable,Dreamer{

}
```

▼ Default and Static Methods

- *Beginning in Java 8*, interfaces can have *default* (different than access modifiers) and ***static***
- Default and static methods have ***a body*** like a regular method.
- How can I reach a static method inside an Interface? (className(interfaceName).staticMethod)
- How can I reach default method inside Interface? (with the object referance)

▼ Abstract Class VS Interface

Abstract classes and interfaces can both be used to specify common behavior of objects.

▼ How do you decide whether to use an interface or a class?

- In general, a strong is-a relationship that clearly describes a parent-child relationship should be modeled using abstract classes.
- A weak is-a relationship, also known as an is-kind-of relationship, indicates that an object possesses a certain property. A weak is-a relationship can be modeled using interfaces.

▼ Differences of both in a single table

	Variables	*Constructors*	*Methods*
Abstract class	No restrictions.	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be `public static final`.	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

▼ Differences of both in a Verbal Manner

- ***Abstract classes and Interfaces are used to achieve abstraction in Java***
- ***We cannot instantiate abstract classes and interfaces***
- We use abstract classes for setting foundation for sub classes. It is normally a general idea
- Interfaces are used to add a feature to classes by providing abstract methods
- Class can ***extends*** ONE abstract class
- Class can ***implement*** MULTIPLE interfaces