OOP with Java

Enumerated data types

* Creating your own data type with its specific own data values that it can accept by specifing them when create the enum data type, by using the special class enum.

* using enum keyword.

enum dataTypeName

* In the same sentence you have to determine the data values of this enum type. by

enum typeName { dataValue1, value2, ... };

separation between every two data values by the comma (,).

Five Apple

After that you can use this enum type you created to declare data fields and variables everywhere in the program.

Example:

type field/variable Name ;

enum type

* when you want assign a data value from the ones you determined for this enum type, it will be like:

type IdentifierName = enumTypeName.dataValue;

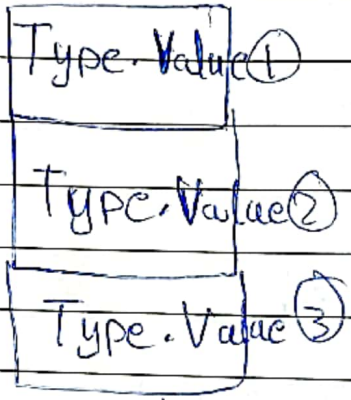
type

Type.dataValue;

All data values of the enum type are called enum consts values.

And these enum consts values are consecutive allocated Objects in the memory.

enum consts values in the memory



and each one of them has its own index.

↓

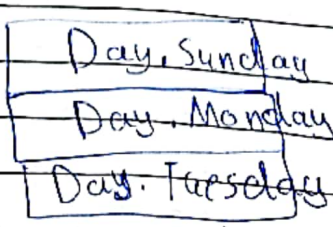
they're like the array.

Example:

```
enum Day { Sunday, Monday, Tuesday, Wednesday };
```

↓ → enum consts

```
Day day = Day.Sunday;
```



There 4 methods we use with enum types:

① toString(): returns the value or the name inside the calling const.

② ordinal(): returns the index of the calling const. (position of the const in the enum).

③ compareTo(): accepts an object of the enum type and returns: ① a positive value if the calling const (of this method) > index (ordinal) > argument const index (ordinal).

② a negative value if the calling const ordinal (index) < argument ordinal (index).

③ zero if the calling const ordinal (index) == the argument ordinal (index).

④ equal(): accepts an object of enum type. returns boolean value:

① true: if the calling const == argument. same ordinal.

② else (otherwise) false.

Five Apple

OOP with Java

Exception handling

by two blocks:

① try{}: Put inside it the code it could cause a run-time error (Exception)

② when the Exception is detected Java throws an object of its type (class) and the catch block receives it by argument (object of the same type or the base class type)

③ catch(){}: object of the Exception class type.

here is the handling of the Exception by printing the Exception error message. By getMessage() method

object of the catch(): from the argument access it by

Exceptions

* Each Exception is a class.

* All of Exception classes are Subclasses of one Super class (base class) which is (Exception).

* If you don't know the specific Exception that will be thrown to the catch() you can receive with a general Exception type object (Object of type Super class (Exception)) because each Exception class inherits from the Super-class Exception (each exception class is a Subclass of the Super class Exception).

So, each exception object is a Exception (every exception class is A Exception class or type).

Five Apple

as a string.
the error message
it returns
getMessage()

Example:

Prgrm Execute

```
try {
}
catch(Exception Error) {
    System.out.println(Error.getMessage());
}
```

Error.getMessage()

the program doesn't crash and it continues to execute the rest of the code after the catch() block (transfer control).

OOP with Java

ArrayList

→ to use it you have to import the ArrayList from package java.util.

import java.util.ArrayList;

Is an array of any datatype ~~or~~ even array of objects of datatype of a specific class, but it has feature more than the normal array.

* you can add to it after determining its size and you can remove elements from it.

* in the other hand,

It has special methods:

1) add() : it adds an element in the end of array. takes an argument (to add it to the array)

② remove(int index): takes an integer argument which represents the index of the element you want to remove from the ArrayList.

③ size(): It returns the size of the ArrayList.

④ get()

④ get(int index): It returns the value of the index.

⑤ To print all elements in the ArrayList use: System.out.println(ArrayListName); overloaded method — use the Name of the ArrayList.

⑥ add(int index, value): It adds the value in the specific index in the ArrayList.

~~rem~~

⇒ set(int index, value) : to change the value of an item at specific index in the ArrayList.

* to declare ArrayList:

~~ArrayList<datatype> nameLis~~

ArrayList<datatype> nameList = new ArrayList<datatype>();

you can use

No args constructor ①

parametrized constructor ②

It sets the size of the List to default size value (10);

It sets the size of the List to the integer value you passed.

Five Apple

OOP with Java

Polymorphism

① abstract classes

1 * It cannot be intantiated (you cannot declare an instance from it (object of its type).

~~* It can contain~~

2 * It may contain non-abstract methods and abstract methods.

3 * you determine whether the method is abstract, or not. By the keyword abstract

4 * abstract methods cannot have implementation and you're forced to

override these abstract methods in each Subclass that inherits from this super abstract class.

(*) If the class contains at least one abstract method, the class will be abstract class, and you're forced to use the "abstract" keyword in the class prototype (to define the class as abstract).

(*) You're forced to Override the abstract methods (of the abstract Super class) in each Subclass of this Super class.

(*) Java doesn't support multiple Inheritance, it only supports single Inheritance.

21 Interf

2 interfaces

1 * they're similar to the abstract classes but they differ at some points.

2 * they connot be instantiated.

differences:

3 * they cannot contain any method with implementation (non-abstract) methods. They only contain abstract methods.

4 * You don't use "abstract" keyword in the prototypes of the methods inside the interface. You just write the method prototype and a(;)

semicolon after that without writing the implementation of the method.

~~(5) (*) The methods of~~

(5) (*) You're forced to Override ~~the~~ or implement the methods of the interface in the class that implements this interface.

keyword

(6) (*) The classe implements the interface ~~not~~ but don't inherit (extends) it.

~~(4) (*) Interfaces inherit from~~

(7) (*) A class can implement more than one interface: Example:

public class A implements Interface1, Int2,

this class separation between the interfaces

implements

FiveApple

Interfaces

* An interface can inherit (extends) from another interface, and the class the class who implements the (Sub) interface (the one that extends) and inherits from the other one) is forced to implement and override the methods of the two interfaces.

* Implementing a method of an interface without @override notation (@Override).

* Implementing this method by writing its prototype and its implementation inside the body.

methods → type of the Object. (allocated obj in the memory)

data fields/attributes or → type of the reference variable. (which the reference variable points to).

(3) Polymorphism with attributes and methods.

~~polymorphism works with methods, but it doesn't work with attributes.~~

polymorphism with methods → depends on the data type

polymorphism with attributes → depends on the Object.

~~Polymorphism~~ the data type of the data fields → reference variable

* Polymorphism: Is having a reference variable of type Superclass and it points to an object in the memory of type Subclass.

~~* Calling the methods by the reference variable~~

* Calling methods exists in both classes by the reference variable (of type Superclass) who points to the Object (of Subclass) in the end.

(Subclass) methods will be called and executed.

method

Five Apple

* The

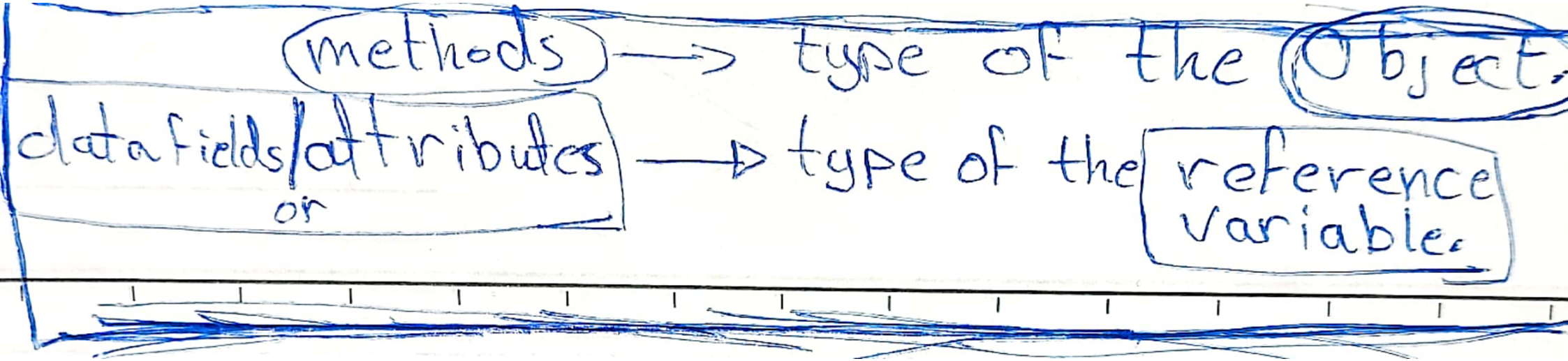
* Calling the data fields (attributes) by the reference variable (of type Superclass) which point to an Object (of type Subclass). In the end the data field of the reference variable (Superclass) will be called.

In Conclusion:

* methods in polymorphis depends on the type of the object.

* ~~data~~

* data fields depend on the type of the reference variable.



Polymorphism: Is having a reference variable of type Super class and it points to an object in the memory of type Subclass.

methods → type of the Object.

data fields/attributes or → type of the reference variable.

Callocated obj in the memory (which the refer variable points to)

3 Polymorphism with attribute and methods.

polymorphism works with methods, but it doesn't work with attributes.

polymorphism with methods → depends on the data type of the Object.

polymorphism with attributes → dedends on the data type of the data fields → reference variable

Polymorphism: Is having a reference variable of type superclass and it points to an object in the memory of type subclass.

polymorphism → with attributes → depends on the data type of the data fields → reference variable

Polymorphism

* Polymorphism: Is having a reference variable of type Super class and it points to an object in the memory of type Subclass.

* Calling the methods by the reference variable

* Calling methods exists in both classes by the reference variable (of type Superclass) who points to the Object (of Sub class)

(Subclass) methods will be called and executed.

in the end the Object

Five Apples

by the reference variable (of type Superclass) which point to an Object (of type Subclass). In the end the data field of the reference variable (Superclass) will be called.

In Conclusion:

* methods in polymorphis depends on the type of the object.
* ~~data~~
* data fields depend on the type of the reference variable.

Inheritance

Polymorphism and Interfaces

mandatory = obligatory (forced to do so).

mandatory and optional Overriding and implementing

3 cases (2 cases mandatory) (1 case optional)

Mandatory overriding and implementing.

3 cases

Optional overriding

Super abstract class implements an Interface and this Interface has a method inside it. And the super abstract class doesn't implement this method inside the super class. ~~Here the Sub~~ Here each Subclass of this Super class forced to implement this method on behalf of the Super class.

Mandatory implemeting //

Super class has an abstract method inside it. Here each Subclass of this Super class is forced to override this method.

~~This (1)~~

Mandatory Overriding.

Note: This abstract method inside super class made the class an abstract class.

Super class has a method inside it and its implementation (the method is not abstract method). And you want to give it different implementation in each Subclass. So you can override it in each Subclass of this Superclass.

Java Access modifiers

4 Types: 1. private keyword
(No-keyword) 2. Default → by Not writing or
3. protected keyword
4. public keyword

by Not specifying the access modifier.

① private

X

Not in same class (outside the same class) → Not accessible X

Same class → accessible ✓

② Default

X

Different package → X Not accessible

Same package (inside the same package) → accessible ✓

③ protected

X

Not a subclass → Different package → X Not accessible

subclass (subclass of our class that contains this data field) → accessible ✓

Same package → accessible ✓

④ public

accessible anywhere ✓