VARIABLE

In java variables are categorised on 2 bases

1. Based on declaration type

* If variable is declared using dataTypes like(int,char) then it will be called Primitive type variable.
* Primitive variable will store value.

If variable is declared using javaType like ClassName, InterfaceName, EnumName then it will be called Non primitive variable or reference variable.

Non primitive variable will store address.

1. Based on Scope

Local Variable – variable which are declared inside the function is called local variable.

There scope is limited to that function only.

Member variable – variable which are declared outside the function and inside the class is known as member variable.

There scope is depends on access specifier.

**class**

* class is a blueprint of an object.
* class will have states and behavior of an object.
* class is not a real world entity that why it will not consume any space in the memory.

**object**

* object is a instance of a class.
* Every object will have its unique states and behaviour.
* Object is a real world entity that why it will consume space in the memory.

access specifier or access modifier – public , private , protected , default

non-access modifier are – static, final , synchronized , native

**Accessing Static members And Non-Static Members of a class**

**From same class**

* From static function we can access static members of same class directly without using className.
* From static function we cannot access non-static members of same class through object reference.

* From non-static function we can access static or non-static members of same class directly.

**From other class**

Rules for static function and non-static function are same i.e

* Static members has to call by the class name.
* And non-static members has to call by the object reference.

**final Keyword with variable**

If we use final keyword with variable then it means

1. that variable will not get default initialization.
2. that variable value is fixed throughout the program.

**Intialization Rules**

final keyword with local variable – Initialize at the time of declaration only.

final keyword with static member variable

* Intialize at the time of declaration
* Intialize through static block

final keyword with non-static member variable

* Initialize at the time of declaration
* Initialize through non-static block
* Initialize through constructor

**final keyword with class**

Then that class methods cannot be inherited means it cannot become super-class, but it can be a sub-class.

Reason

final means which is constant which can’t be changed(in java) so if a class become super class then its properties can be overridden in sub-class, so that why final class cannot become super class.

**final keyword with function**

That function can be inherited but cannot override.

**package**

* Package is a collection of java program.
* Package can have sub-package also.

package syntax- package packageName

Rules

* package declaration should be the first line of our java program.
* One program can have only one package.

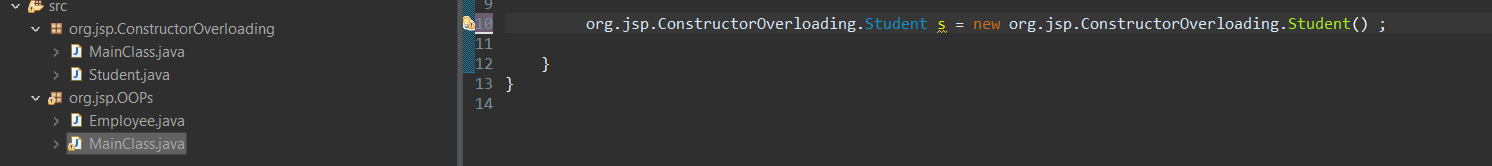
Q- Can we have two class with same name in one package ?

1. No because it will leads to confusion.

Q- Can we make object of a class which is present in some other package ?

Yes

1. Import that class and then make object.
2. If you don’t want to import then, use fully qualified class name while making object of that class. For ex - inbox.A a1 = new inbox.A() ;



FullyQualifiedClassName means packagename.ClassName

**import**

import is used to import a class which is present in some other package to our program(source code).

Syntax - import fullyQualifiedClassName

Rule

Import statement should be written after the package declaration.

For ex- java.util.Scanner , java.util.\*

**Inheritence**

Constructor Chaining means when two or more constructor are running to initialize an object.

Constructor Chaining is achieved by super() calling statement or this() calling statement.

super() calling statement is used to call the super class constructor.

this() is used to call the constructor of same class.

Rules for this() calling statement

1. this() calling statement should be the first line of constructor
2. one constructor should be without this() calling statement.(otherwise ifinite loop will happen which results to stack overflow)

Whenever we write inheritance program, super-class constructor will be called compulsory from sub-class constructor.

From sub-class constructor either implicit call will happen or explicit call will happen to super class constructor.

Implicit call will be done by the compiler only if

1. Programmer has not made any expilicit call.
2. super-class has one no argument constructor.

And any of the condition is not fulfilled then it is mandatory to explicitly call the super-class constructor using super calling statement.

Rules for super() calling statement

1. super() calling statement should be the first line of our sub-class constructor.
2. super() calling statement must be called from sub-class constructor body only.
3. Only one super() calling statement is allowed in one constructor.

Note- See the program in eclipse for better understanding

**Members we can’t inherit & override**

1. private
2. constructor
3. static

These members we can’t override because for overriding, inheritance is mandatory and these members can’t be inherited only.

Method-overloading real life example

1)online payments – upi,netbanking,debit card,credit card

2)flight search- search by destination , search by pnr

Method-overriding example

Note- super keyword will hold the reference of immediate super class of an object. User super keyword we can access super class properties.

**interface**

Interface is a java type definition which is used to provide standardization and abstraction.It tells what to do but not how do it.

interface members

static variable YES

Non-static variable NO (because all variables are by default public ,static, final)

Constructor NO (constructor is used to intitalize the non-static variables but in interface we can’t write non-static variable, so no use of constructor)

Blocks NO

static Methods(abstract) NO (not only in interface but java language only static abstract method are not allowed because we access static method with class name directly if they are doing nothing there is not point of making them)

static methods(concrete) YES (from java 8 )

non-static methods(abstract) YES (from starting version only)

non-static methods(concrete) YES (from java 8 but one condition use default keyword while declaring non-static concrete methods)

About interface

* Interface is by default abstract, so no need of using abstract keyword while declaring interface and while declaring abstract method inside interface..
* Interface is abstract so we can’t make object of interface.(you can declare reference variable )
* All members of interface are by default public.(until declared default)

About interface variable

* all variables in interface is by default public, static and final.
* As all variable are final so you have to initialize them at the time of declaration only or through static block.

Generally interface is used to create abstract methods, but we can create static concrete methods and non-static concrete methods (from java 8) but only condition is non-static concrete method should be declared using default keyword.

Note – Rest questions and code read from notebook

Before reading Typecasting understand this

**Type-matching**

**(type of) L.H.S == (type of) R.H.S ….. type of L.H.S must be equal to R.H.S**

**Data-TypeCasting**

When we convert one datatype into another datatype is called data-typecasting.

Data-type casting is of 2 types

1. Data-widening
2. Data-narrowing

Data-widening

* It means when we convert lower data-type(in-size) into higher datatype is called data-widening.

For ex- double a = 5 ;

Here int value is converted to double… lower to higher(4 byte data going in 8 bytes variable)

* So here no data-loss happening, so it will be done by the compiler implicitly that why it is called implicit type-casting.(we can do it explicitly also)

Data-Narrowing

* It means when we convert higher datatype into lower datatype is called data-narrowing.

For ex- int a = (int)5.68 ;

Here double value is converted to int…..higher to lower(8 bytes data going in 4 bytes variable)

* So here data loss will happen, so progarmmer has to do it explicitly.(programmer has to tell to compiler that he is ready to lose some data).

**Class-TypeCasting**

* When we convert one class type to another classtype is called class-typecasting.
* For class-casting “is-a” relationship is mandatory between the classes, otherwise ClassCastException will come.

Class casting is of two types

1)Upcasting

2)Downcasting

Upcasting

* When we convert sub-class type to super class type is known as Upcasting.
* It is done to access super-class general properties. Here sub-class object will behave like super-class object.
* Upcasting is done by the compiler implicitly.(We can also do it explicitly)

For ex- Demo1 d1 = new Demo2() ;

//Here Demo2 which is a sub-class object is upcasted to super class object of Demo1, implicit typecasting is happening by the compiler.

DownCasting

* When we convert super-class type to sub-class type is known as DownCasting.
* It is done to access sub-class specific properties.
* Downcasting must be done explicitly by the programmer.

For ex- Demo2 d2 = (Demo2) d1 ;

//Here d1 which is upcasted to super-class object is again downcasted to sub-class object.

Note- Only that object is downcasted, which was once upcasted, if you try to downcast super-class object directly to sub-class object, JVM will throw ClassCastException because super-class will never have the sub-class properties.

**Implementation class**

Implementation class means a class which implements(provide body to) all the methods of the interface.

**Abstraction**

Hiding the implementation detail and showing the functionality to the user is called Abstraction.

How to achieve Abstraction

1. Define functionalities(methods) in interface.
2. Make implementation classes for these functionalities.
3. And then make object of implementation class and store it in the interface type reference variable. And then through interface type reference variable we will access the functionalities.

Now even though we are calling the methods(functionalities) through interface type reference variable, but binding of method body with method declaration is based on object. So when we created the object of implementation class, method declaration will get binded to method body. So same method declaration is present in interface also, which are already binded with method body while creating implementation class object. So this is why when you call interface methods, method body, present in implementation class get executed.

Note – writing program for run-time polymorphism and abstraction both have same logic

In both first you do inheritance then method-overriding then upcasting but difference is, in run-time polymorphism first should be super class and rest two will be sub-classes, in abstraction first should be interface(because you have to hide implementation and show only functionality) and rest two will be implementation classes.

For Program see eclipse

|  |  |
| --- | --- |
| abstract class | interface |
| 1. We have to use abstract keyword while declaring abstract class and abstract method. | * interface is by-default abstract no need to use abstract keyword while declaring interface and abstract method. |
| 1. We can make any type of non-static concrete method here. | * We can make only default non-static concrete method here. |
| 1. No restrictions on data members. | * Data members must be public, static and final. |
| 1. Inherit abstract class property through extends keyword. | * Inherit interface property through implements keywords. |
| 1. Multiple inheritance is not possible | * Multiple inheritance is possible. |
| 1. Partial abstract body, 100% abstraction is not possible. | * Fully abstract body, 100% abstraction will be achieved. |

Q- Why abstract class is called partial abstract body ?

Because it is a class so it will be inheriting concrete method of Object class that why it is called partial abstract body.

Q- Why we cannot achieve 100% abstraction through abstract class?

Because Abstract class is a class, and every class in java has a super-class called Object class, so it will be inheriting the concrete methods of Object class. So that’s why we say through abstract class you cannot achieve 100% abstraction.