We can define a group of named constants.

EX: [1]enum month {jan,feb,mar,apr….dec; }

[2]enum day {sun,mon….sat}

[3]enum day {sun(0),mon(1)….sat(6)}

🡺 ; is optional, when we declare only constants.

🡺Sun ,mon….sat are the elementa or constantsand 0,1….6 are the parameters of valuesresp.

🡺1:Every enum constant is by default public static final.

🡺2:Every enum constant is implemented by using class concept.

🡺3:Every enum constant represents object of the class.

The above code[2] is represented as below by jvm and saves as day.class

Class day{ //point 2

Public static final sun=new day(); //point 1,3

:

:

Public static final sat=new day();

}

Accessing enum constants:

enum fish{star,gold,guppy;}

class demo{

public static void main(String [] args)

{ fish f1=fish.star;

S.o.p(f1); }}

o/p:star

Accessing enum constants and its values:

When custom constructor’s is declared, all element’s declarations must match that custom constructor’s.

enum Bear

{

KF(100),RC(150),FO(200),TEN(),Two(10,20);//elements declaration

int price,x,y;//instance variables

Bear(int x,int y) //custom constructor

{

this.x=x;

this.y=y;

price=x+y;

}

Bear(int price)//custom constructor

{

this.price=price;

}

Bear()//custom constructor

{

price=500;

}

public int getPrice()

{

return price;

}

}

class EnumDemo1

{

public static void main(String[] args)

{

Bear b1=Bear.KF;

System.out.println(b1+"--"+b1.price);

Bear b2[]=Bear.values();

for(Bear b3:b2)//enhanced for loop

{

System.out.println(b3+"----"+b3.getPrice());

}

}

}

o/p:

KF--100

KF----100

RC----150

FO----200

TEN----500

Two----30

TEN,TEN()🡺match with no parameter custom constructor.

KF🡺match with single parameter custom const.

RC🡺match with single parameter cusom const.

Two🡺match with two parameter const.

TEN ,TEN() are constants or elements

TEN🡺constant with no value,TEN(10)🡺constant with value.

TEN,TEN()🡺both will go to the no parameter custom constrctor.

KF(100)🡺go to the single parameterized constructor.

TWO(10,20)🡺go to the double parameterized constructor.

We can also use enum constrant , as argument to switch statement.

EX: enum Bear{A,B,C}

Class Demo{

Public Static void main(String[] args)

{

Bear b1=Bear.A;

Switch(b1)

{

Case A:System.out.println(“A”);break;

Case B:System.out.println(“B”);break;

Case C:Syste.out.println(“C”);break;

}

}

}

//when ever we are passing enum constant as argument to switch statement, every case label should be enum constant otherwise we will get compile time error saying “unqualified enum constant”.

Every enum in java is directly child of java.lang.Enum class

🡺every user defined enum class is already (by default)extend java.lang.Enum class , so there is no chance of extending any otherclass(enum & normal class).Hence inheritance concept is not applicable for enum.

For example:

enum a{}

enum b extends a{} //here we ll get compile time error, because enum does not support inheritance.

🡺But enum can implement any no of interfaces.

EX: public interface AB{

Public void M1();

Public void M2();

}

enum a implements AB{

public void M1()

{System.out.println(“M1”);}

Public void M2()

{System.out.println(“M2”)}

}

🡺public,default and strictfp are the only applicable for enum if we are applying any other modifiers to enum then we will get compile time error.

🡺Inside enum we can define instane variables,instance methods & static variables ,static methods.

🡺we can declare enum outside and inside of the class but not out side of the method.

EX:[1]Enum A{}

Class Demo

{

//valid

}

EX[2]:

Class Demo{

enum A{

//valid

}

//

}

EX:[3]

Class Demo{

Public void M1()

{

Enum A{

//invalid

}

}

}

Java.lang.Enum

🡺it acts as a base class for all enum(user defined here Bear,fish) in java

🡺Every enum has a direct sub class of java.lang.Enum.

🡺Enum is an abstract class and direct child of object.

🡺Enum class implements Serializable and comparator interface.

🡺Syntax for java.lang.Enum declared in java docs is

Public abstract class Enum extends object implements Serializable,Comparable{

//

//

}

🡺In short, Enum can have constructor, methods and inbuilt objects.

🡺our enum class always by default extends java.lang.Enum class

🡺Methods of Enum:

Public Static java.lang.Enum[] values();

This method returns list of all constant defined in enum.

Public integer ordinal();

With in enum the order of constants are important , it can be specified by ordinal values.

This method returns ordinal(index) values of enum constant.

EX:enum Bear {EF,FO,TEN,RC}

Class Test {

Public Static void main(String[]args){

Bear[] b=Bear.values();

for(int i=0;i<b.length;i++)

System.out.println(b[i]+”--“+b[i].ordinal());

}

}

Output: (constant name--ordinal value of element)

EF--0

F--1

TE--2

RC--3

Another example of enum:

public enum UserStatus {

PENDING("P"), ACTIVE("A"), INACTIVE("I"), DELETED("D");

private String statusCode;

private UserStatus(String s) {

statusCode = s;

}

public String getStatusCode()

{return statusCode;

}}

public class test {

public static void main(String[] args) {

System.out.println(UserStatus.ACTIVE.getStatusCode());

}

}

**Speciality of enum:**

**🡺I**n enum ,we can define methods ,constructors,instance variables etc in addition to group of constants,which may not be possible in old languages enum.

🡺we can also define main() method inside of the enum and we can invoke enum class directly from the command prompt.

EX: enum bear{KF,RC,FO,TEN

Public static void main(String[] args)

{

System.out.println(“enum main method”);

}

}

🡺we should save the file as bear.java

🡺java bear.java

🡺java bear

o/p:enum main method

🡺At the time of enum class loading, all enum constants will be created and executing the corredponding constructors.

🡺we cannot create the enum object explicitly and hence we cannot invoke enum constructors explicity

i.e., Bear b1=new Bear(); //gives compilation error

🡺Inside enum we can define instance ,static variables, & instance, static methods but not abstract methods.

**Enum that overrides toString method**. A semicolon after the last element is required to be able to compile it.

enum color {

WHITE, BLACK, RED, YELLOW, BLUE; //; is required here.

public String toString() {

//only capitalize the first letter

String s = super.toString();

return s.substring(0, 1) + s.substring(1).toLowerCase();

}

}

class enumdemo2

{

public static void main(String[] args)

{

color[] c1=color.values();

for(int i=0;i<c1.length;i++)

System.out.println(c1[i]+"--"+c1[i].ordinal());

}

}

o/p:

White--0

Black--1

Red--2

Yellow--3

Blue—4

**Enum with additional fields and custom constructor**. Enum constructors must be either private or package default, and protected or public access modifier is not allowed. When custom constructor is declared, all elements declaration must match that constructor.

publicenumColor {

 WHITE(21), BLACK(22), RED(23), YELLOW(24), BLUE(25);

 privateintcode;

 privateColor(intc) {

   code = c;

 }

 publicintgetCode() {

   returncode;

 }

**Enum that implements interfaces**. Enum can implement any interfaces. All enum types implicitly implements java.io.Serializable, andjava.lang.Comparable

enum color implements Runnable {

WHITE, BLACK, RED, YELLOW, BLUE;

public void run()

{

System.out.println("name()=" + name() +", toString()=" + toString());

}

}

class EnumDemo3

{

public static void main(String[] args)

{

for(color c : color.values()) {

c.run();

}

}

}

//same as above 3 lines for(Runnable r : color.values()) {

// r.run();

//}

A more complete example with custom fields, constructors, getters, lookup method, and even a main method for quick testing:

|  |
| --- |
| importjava.util.HashMap;  importjava.util.Map;    publicenumStatus {      PASSED(1, "Passed", "The test has passed."),      FAILED(-1, "Failed", "The test was executed but failed."),      DID\_NOT\_RUN(0, "Did not run", "The test did not start.");        privateintcode;      privateString label;      privateString description;  //A mapping between the integer code and its corresponding Status to facilitate lookup  //by code.        privatestaticMap<Integer, Status> codeToStatusMapping;        privateStatus(intcode, String label, String description) {          this.code = code;          this.label = label;          this.description = description;      }        publicstaticStatus getStatus(inti) {          if(codeToStatusMapping == null) {              initMapping();          }          Status result = null;          for(Status s : values()) {              result = codeToStatusMapping.get(i);          }          returnresult;      }        privatestaticvoidinitMapping() {          codeToStatusMapping = newHashMap<Integer, Status>();          for(Status s : values()) {              codeToStatusMapping.put(s.code, s);          }      }        publicintgetCode() {          returncode;      }        publicString getLabel() {          returnlabel;      }        publicString getDescription() {          returndescription;      }        @Override      publicString toString() {          finalStringBuilder sb = newStringBuilder();          sb.append("Status");          sb.append("{code=").append(code);          sb.append(", label='").append(label).append('\'');          sb.append(", description='").append(description).append('\'');          sb.append('}');          returnsb.toString();      }        publicstaticvoidmain(String[] args) {          System.out.println(Status.PASSED);          System.out.println(Status.getStatus(-1));      }  } |

To run the above example:

|  |  |
| --- | --- |
|  | java Status    Status{code=1, label='Passed', description='The test has passed.'}  Status{code=-1, label='Failed', description='The test was executed but failed.'} |