**Enum**

1. An **enum** is a data type (reference type or special java type) which defines fixed set of constants.
2. An enum is NOT a String or an int; an enum constant's type is the enum type. For example, SUMMER and FALL are of the enum type Season.
3. An enum can be declared outside or inside a class, but NOT in a method.
4. An enum declared outside a class must NOT be marked static, final, abstract, protected, or private.
5. Enums can contain constructors, methods, variables, and constant class bodies.
6. enumconstants can send arguments to the enumconstructor, using the syntax BIG(8), where the int literal 8 is passed to the enumconstructor.
7. enumconstructors can have arguments, and can be overloaded.
8. enumconstructors can NEVER be invoked directly in code. They are always called automatically when an enumis initialized.
9. The semicolon at the end of an enumdeclaration is optional. These are legal:

enum Foo { ONE, TWO, THREE}

enum Foo { ONE, TWO, THREE};

1. MyEnum.values() returns an array of MyEnum's values.

In Detail:

1. Enum is used to define constant with its symbolic name value.
2. enum keyword is used to declare the enum type.
3. Enum can be declared as their own separate class or inside the class.
4. Enum can’t be declared inside a method.
5. Enum in its own separate class:

**publicenum** Day {

*MON*,

*TUE*,

*WED*,

*THRU*,

*FRI*,

*SAT*,

*SUN*

}

* We can save this file as Day.java file name.
* enum declared outside as a separate class can’t be private, protected, static, final, abstract.

1. How to use enum in another class.

**publicclass**DayManager {

**private** Day d;// declare a reference of enum type

**public** Day getD() {

**return**d;

}

**publicvoid**setD(Day d) {

**this**.d = d;

}

**publicstaticvoid** main(String[] args) {

//Day d1 = new Day();// compile time error => can't instantiate the type Day

DayManagerdm = **new**DayManager();

dm.setD(Day.*SUN*);// only pass the enum type data

//dm.setD(1);//compile time error => The method setD(Day) in the type DayManager is not applicable for the arguments (int)

Day d2 = dm.getD(); // get only enum type

dm.d = Day.*MON*;

dm.show(d2);

}

//enum can be used in the switch case condition

**privatevoid** show(Day d2) {

**switch**(d2){

**case***MON*: System.*out*.println(d2);**break**;

**case***TUE*: System.*out*.println(d2);**break**;

**case***WED*: System.*out*.println(d2);**break**;

**case***THRU*: System.*out*.println(d2);**break**;

**case***FRI*: System.*out*.println(d2);**break**;

**case***SAT*: System.*out*.println(d2);**break**;

**case***SUN*: System.*out*.println(d2);**break**;

**default**: System.*out*.println("Error.......");

}

}

}

1. Declare enum inside a class.

**package**com.test;

**publicclass**DayDisplay {

**enum** Day {

*MON*, *TUE*, *WED*, *THU*, *FRI*, *SAT*, *SUN*

} // ; semicolon is optional

**static** Day *d1*;

**publicstaticvoid** main(String[] args) {

//enumMonth{JAN,FEB} // compile time error: The member enum Month can only be defined inside a top-level class or interface

Day d2 = Day.*MON*;

System.*out*.println(Day.*MON*);

System.*out*.println(d2);

System.*out*.println(d2.toString());

System.*out*.println(Day.*values*()); //[Lcom.test.DayDisplay$Day;@1cc2ea3f

Day [] val = Day.*values*();

**for**(Day d : val){

System.*out*.println("d.name() = "+ d.name()+ " d.ordinal() = " + d.ordinal() + " d.toString() = "+ d.toString());

}

*d1* = Day.*FRI*;

System.*out*.println(*d1*);

}

}

**Output:**

MON

MON

MON

[Lcom.test.DayDisplay$Day;@1cc2ea3f

d.name() = MON d.ordinal() = 0 d.toString() = MON

d.name() = TUE d.ordinal() = 1 d.toString() = TUE

d.name() = WED d.ordinal() = 2 d.toString() = WED

d.name() = THU d.ordinal() = 3 d.toString() = THU

d.name() = FRI d.ordinal() = 4 d.toString() = FRI

d.name() = SAT d.ordinal() = 5 d.toString() = SAT

d.name() = SUN d.ordinal() = 6 d.toString() = SUN

FRI

1. Conceptually enum means:

**enum** Day {

*MON*, *TUE*, *WED*, *THU*, *FRI*, *SAT*, *SUN*

}

Means:

class Day {

public String name;

publicint ordinal;

public static final Day MON = new Day(“MON”,0);

public static final Day TUE = new Day(“TUE”,1);

public static final Day WED = new Day(“WED”,2);

public static final Day THU = new Day(“THU”,3);

public static final Day FRI = new Day(“FRI”,4);

public static final Day SAT = new Day(“SAT”,5);

public static final Day SUN = new Day(“SUN”,6);

public Day(String enum\_name, int index){

this.name = enum\_name;

this.ordinal = index;

}

public string toString(){

returnname;

}

}

* enum is just like a class.
* Each enum constant is the public static final instance variable of the class of same class type.
* These instance variables are initialized by the instance of the class by passing below two arguments.

1. Name of the enum (it’ll be same as the enum constant name)
2. Index of the enum (it’ll be same as the enum constant position start from 0 )

* When this class loads in the memory the instance variable will get memory on heap and it’ll be initialized by given value.

1. enum can declare constructors and other class members (method, variables, constant’s class body)

* enum constant must be declared before any other declaration.
* enum constants send arguments to the enum constructor as like below:

MON(1, “it’s Monday today.”)

* enum constructors can be overloaded.
* enum constructor can never be invoked directly from the code by the developer.
* enum constructors are always called automatically by JVM, when enum constant is initialized.

(enum constant is initialized at enum loading time because enum constant is public static final instance variable of enum)

1. enum constant’s class body:

* enum can have constant’s class body.
* enum constant’s class body is specific to a particular enum constant.
* It’s enclosed inside { }.
* “{” starts immediately after the enum constant or after the argument constructor.
* The class bodies can contain member’s declaration and definition.
* Constructor, abstract method and static methods can’t be inside a constant’s class body.
* Instance methods declared in constant-specific class bodies areonly accessible if they override methods in the enclosing enumsupertype.
* If enum type contains any abstract method then each enum constant has to override that method inside the constant’s class body.
* **Note:** The constructor for an enum type must be package-private or private access. It automatically creates the constants that are defined at the beginning of the enum body. You cannot invoke an enum constructor yourself.
  + - all interface methods are public and abstract ... so in interface Runnable ( void run ( ) ; } the run method is public and abstract.
    - Perhaps the java designers did the same thing for enum constructors (they are private whether or not you mark them private)

BREAKFAST(7,30) { // (1) Start of constant-specific class body

**publicdouble**mealPrice(Day day) { // (2) Overriding abstract method

...

}

**public** String toString() { // (3) Overriding method from the Enum

**class**

...

}

} // (4) End of constant-specific class body

Example1: Use of enum.

**package**com.test.enumtest;

**enum**MachineState {

*BUSY*, *IDLE*, *BLOCKED*

}

**class** Machine {

**private**MachineStatestate;

**publicvoid**setState(MachineState state) {

**this**.state = state;

}

**public**MachineStategetState() {

**returnthis**.state;

}

}

**publicclass**MachineClient {

**publicstaticvoid** main(String[] args) {

Machine machine = **new**Machine();

machine.setState(MachineState.*IDLE*); // (1) Passed as a value.

//machine.setState(1); // (2) Compile-time error!

//The method setState(MachineState) in the type Machine is not applicable for the arguments (int)

MachineState state = machine.getState(); // (3) Declaring a reference.

System.*out*.println("The machine state is: " + state); // (4) Printing the enum name.

//MachineStatenewState = new MachineState();// (5) Compile-time error!//Cannot instantiate the type MachineState

}

}

Output:

The machine state is: IDLE

Example:Enum with members(constructor, instance variables, instance methods).

**package**com.test.enumtest;

//Filename: Meal.java

**enum** Meal {

*BREAKFAST*(7, 30), *LUNCH*(12, 15), *DINNER*(19, 45); // (1)

// Non-default constructor (2)

Meal(**int**hh, **int** mm) {

**assert** (hh>= 0 &&hh<= 23) : "Illegal hour.";

**assert** (mm >= 0 && mm <= 59) : "Illegal mins.";

**this**.hh = hh;

**this**.mm = mm;

}

// Fields for the meal time: (3)

**privateint**hh;

**privateint**mm;

// Instance methods: (4)

**publicint**getHour() {

**returnthis**.hh;

}

**publicint**getMins() {

**returnthis**.mm;

}

}

// Filename: MealAdministrator.java

**publicclass**MealAdministrator {

**publicstaticvoid** main(String[] args) {

System.*out*.printf(

// (5)

"Please note that no eggs will be served at %s, %02d:%02d.%n",

Meal.*BREAKFAST*, Meal.*BREAKFAST*.getHour(),

Meal.*BREAKFAST*.getMins());

System.*out*.println("Meal times are as follows:");

Meal[] meals = Meal.*values*(); // (6)

**for** (Meal meal : meals)

// (7)

System.*out*.printf("%s served at %02d:%02d%n", meal, meal.getHour(),

meal.getMins());

Meal formalDinner = Meal.*valueOf*("DINNER"); // (8)

System.*out*.printf("Formal dress is required for %s at %02d:%02d.%n",

formalDinner, formalDinner.getHour(), formalDinner.getMins());

}

}

Output:

Please note that no eggs will be served at BREAKFAST, 07:30.

Meal times are as follows:

BREAKFAST served at 07:30

LUNCH served at 12:15

DINNER served at 19:45

Formal dress is required for DINNER at 19:45.

Example:

**package**com.test.enu;

**import**java.util.Arrays;

**publicenum**DailyRoutine {

*WAKEUP* (6,0),

*FRESHENUP*(6f,6.30){

@Override

**public** String show(){

**returnthis**.toString()+ " \t " + start + " to " +end;

}

},

*EXERCISE*(6.30, 7.00){

@Override

**public** String show(){

**returnthis**.toString()+ " \t " + start + " to " +end;

}

},

*BATH* (7.30, 8.00),

*BREAKFAST*(8.00,8.30){

@Override

**public** String show(){

**returnthis**.toString()+ " \t " + start + " to " +end;

}

},

*OFFICE*(9, 18),

*LUNCH* (12.30, 13.30),

*TEA* (5,5.30),

*MEDITATION*(6.30,7.30){

@Override

**public** String show(){

**returnthis**.toString()+ " \t " + start + " to " +end;

}

},

*SKILLIMPROVEMENT*(7.30,8.30){

@Override

**public** String show(){

**returnthis**.toString()+ " " + start + " to " +end;

}

},

*DINNER*(8.30,9.00),

*NIGHTREADING*(9.30,10.30){

@Override

**public** String show(){

**returnthis**.toString()+ " \t " + start + " to " +end;

}

},

*SLEEP*(10.30,6);

**double**start;

**double**end;

DailyRoutine(**double** start, **double** end){

**this**.start = start;

**this**.end = end;

}

**publicstaticvoid** main(String [] args){

//DailyRoutinedr = new DailyRoutine();//Cannot instantiate the type DailyRoutine

System.*out*.println(Arrays.*asList*(DailyRoutine.*values*()));

**for**(DailyRoutine d :DailyRoutine.*values*()){

System.*out*.println(d.show());

}

}

**public**String show(){

**returnthis**.toString()+ " \t\t " + start + " to " +end;

}

}

Output:

[WAKEUP, FRESHENUP, EXERCISE, BATH, BREAKFAST, OFFICE, LUNCH, TEA, MEDITATION, SKILLIMPROVEMENT, DINNER, NIGHTREADING, SLEEP]

WAKEUP 6.0 to 0.0

FRESHENUP 6.0 to 6.3

EXERCISE 6.3 to 7.0

BATH 7.3 to 8.0

BREAKFAST 8.0 to 8.3

OFFICE 9.0 to 18.0

LUNCH 12.3 to 13.3

TEA 5.0 to 5.3

MEDITATION 6.3 to 7.3

SKILLIMPROVEMENT 7.3 to 8.3

DINNER 8.3 to 9.0

NIGHTREADING 9.3 to 10.3

SLEEP 10.3 to 6.0

*All* enum types implicitly have the following static methods, and methods with

these names cannot be declared in an enum type declaration:

The static method values() is called at (6) in Example 3.5 to create an array of enum

constants. This array is traversed in the for(:) loop at (7), printing the information

about each meal. The for(:) loop is discussed in Section 6.3, p. 220.

static*EnumTypeName*[] values()

Returns an array containing the enum constants of this enum type, *in the order*

*they are specified*.

static*EnumTypeName*valueOf(String name)

Returns the enum constant with the specified name. An IllegalArgumentException

is thrown if the specified name does not match the name of an enum constant.

The specified name is *not* qualified with the enum type name.

All enum types are subtypes of the java.lang.Enumclass which provides the default

behavior. All enum types are comparable (Section 15.1, p. 765) and serializable

All enum types inherit the following final methods from the java.lang.Enumclass,

and these methods can therefore not be overridden by an enum type:

protected final Object clone()

An instance of an enum type can*not* be cloned (see Section 10.2, p. 424). The

method throws an CloneNotSupportedException.

finalintcompareTo(E o)

The *natural order* of the enum constants in an enum type is according to their

*ordinal values* (see the ordinal() method below). The compareTo() method in the

Comparable interface is discussed in Section 15.1, p. 765.

finalboolean equals(Object other)

This method returns true if the specified object is equal to this enum constant

(Section 15.1, p. 751).

protected final void finalize()

An enum constant cannot be finalized, because this final method effectively

preventsenum types from implementing their own finalize() method (see

Section 9.4, p. 396).

final Class<E>getDeclaringClass()

This method returns the Class object corresponding to this enumconstant's

enum type (see Section 10.2, p. 424).

finalinthashCode()

This method returns a hash code for this enum constant (see Section 15.1, p. 760).

final String name()

This method returns the name of this enum constant, exactly as declared in its

enum declaration.

finalint ordinal()

This method returns the *ordinal value* of this enum constant (that is, its position

in its enum type declaration). The first enum constant is assigned an ordinal

value of zero. If the ordinal value of an enum constant is less than the ordinal

value of another enum constant of the same enum type, the former occurs

before the latter in the enum type declaration.

protected final Object clone() throws CloneNotSupportedException {

throw new CloneNotSupportedException();

}

/\*\*

\* enum classes cannot have finalize methods.

\*/

protected final void finalize() { }

Thus, for any two enumreferencesmeal1 and meal2, the expression meal1.equals(meal2) and meal1 == meal2

are equivalent.

An enum type cannot be explicitly extended using the extendsclause. An enum

type is implicitly final, unless it contains constant-specific class bodies. If it

declares constant-specific class bodies, it is implicitly extended. No matter what, it

cannot be explicitly declared final.

An **enum** type cannot be *declared***abstract**, regardless of whether each **abstract**

method is overridden in the constant-*specific***class** body of every **enum** constant.

//Like a **class**, *an***enum** can implement interfaces.

**publicinterface**ITimeInfo {

**publicint**getHour();

**publicint**getMins();

}

**publicenum**Meal**implements**ITimeInfo{

// ...

**publicint**getHour() { **returnthis**.hh; }

**publicint**getMins() { **returnthis**.mm; }

}

public abstract class java.lang.Enum extends java.lang.Object implements java.lang.Comparable,java.io.Serializable{

public final java.lang.String name();

public final int ordinal();

protectedjava.lang.Enum(java.lang.String, int);

publicjava.lang.StringtoString();

public final boolean equals(java.lang.Object);

public final inthashCode();

protected final java.lang.Object clone() throws java.lang.CloneNotSupportedException;

public final intcompareTo(java.lang.Enum);

public final java.lang.ClassgetDeclaringClass();

public static java.lang.EnumvalueOf(java.lang.Class, java.lang.String);

protected final void finalize();

publicintcompareTo(java.lang.Object);

}

Questions:

**publicenum** Test13 {

*ATTENTION*("Attention!"), *EYES\_RIGHT*("Eyes right!"), *EYES\_LEFT*("Eyes left!"), *AT\_EASE*("At ease!");

**private** String command;

Test13(String command) {

**this**.command = command;

}

**publicstaticvoid** main(String[] args) {

System.*out*.println(*ATTENTION*); // (1)

System.*out*.println(*AT\_EASE*); // (2)

}

}

Output:

ATTENTION

AT\_EASE

**package**com.test.enu;

**import**java.util.Arrays;

**publicenum** Test13 {

*ONE*(1) {

**public** String toString() {

**return**"LOW";

}

}, // (1)

*TWO*(2), *THREE*(3) {

**public** String toString() {

**return**"NORMAL";

}

}, // (2)

*FOUR*(4), *FIVE*(5) {

**public** String toString() {

**return**"HIGH";

}

}; // (3)

**privateint**pValue;

Test13(**int**pValue) {

**this**.pValue = pValue;

}

**publicstaticvoid** main(String[] args) {

System.*out*.println(Arrays.*toString*(Test13.*values*()));

**for** (Test13 t : Test13.*values*()) {

System.*out*.println(t.name());

}

}

}

Output:

[LOW, TWO, NORMAL, FOUR, HIGH]

ONE

TWO

THREE

FOUR

FIVE

Question: How to use enum for singleton?

Answer:

// Define what the singleton must do.

publicinterfaceMySingleton {

publicvoiddoSomething();

}

privateenumSingletonimplementsMySingleton {

/\*\*

\* The one and only instance of the singleton.

\*

\* By definition as an enum there MUST be only one of these and it is inherently thread-safe.

\*/

INSTANCE {

@Override

publicvoiddoSomething() {

// What it does.

}

};

}

publicstaticMySingletongetInstance() {

returnSingleton.INSTANCE;

}