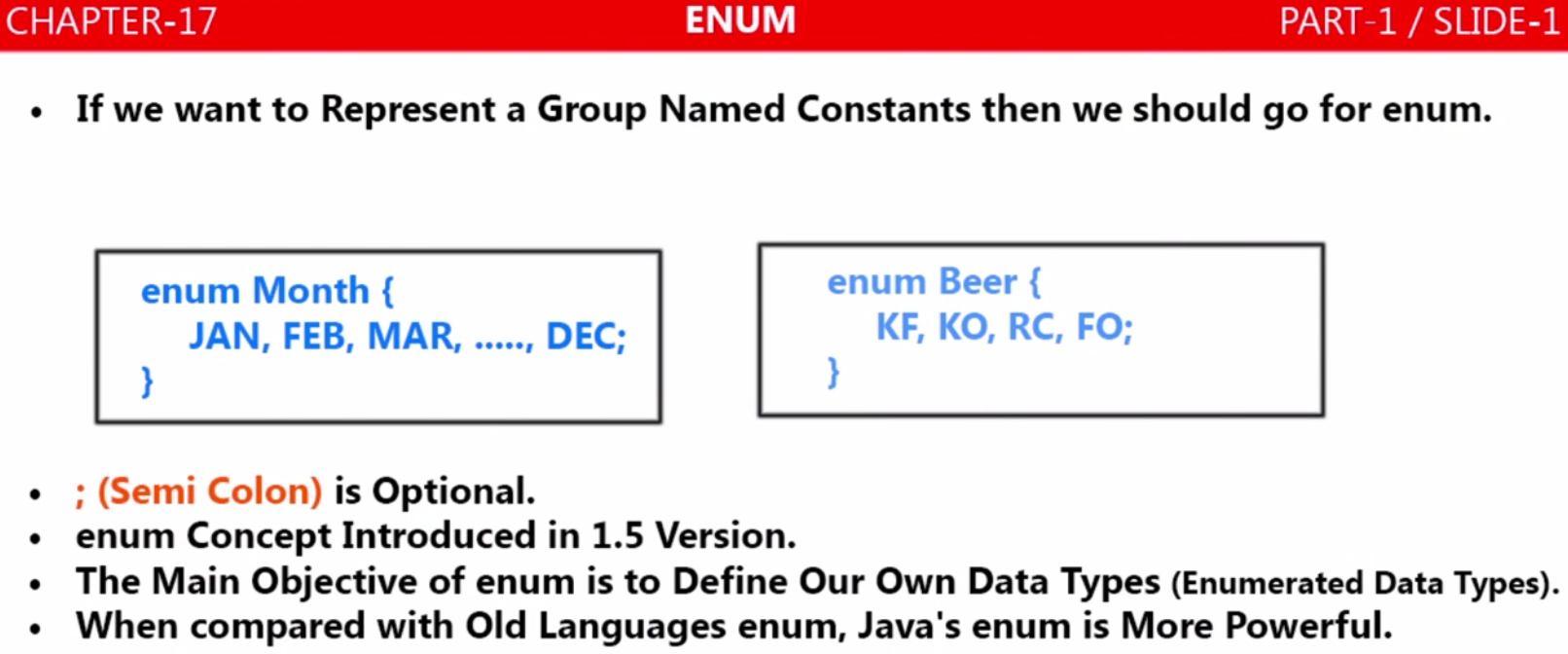
**:Java ENUM:Enumeration**



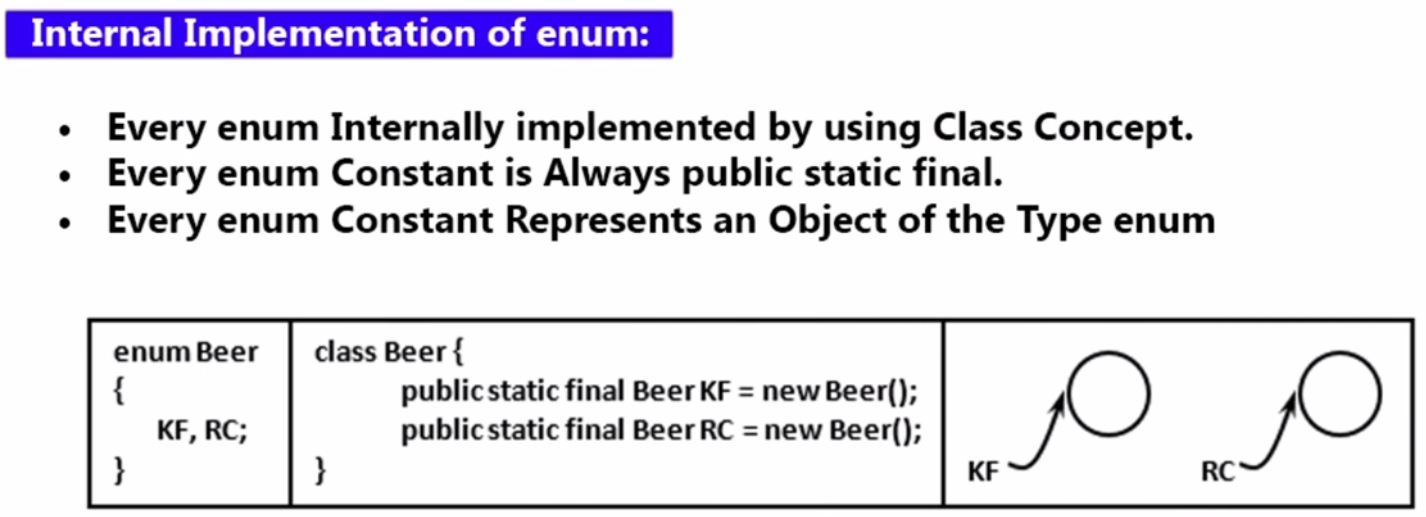
* If we want to represent a group of named constant then we should go for enum
* For example if we want to represent all the month’s name under a single name or all the week’s name under a single name then this thing we can represent with Java’s enum as given below

|  |  |
| --- | --- |
| **public** **enum** Month {  ***JAN***,***FEB***,***MAR***,***APR***,***MAY***,***JUN***,***JUL***,***AUG***,***SEP***,***OCT***,***NOV***,***DEC***;  }  12 values are there under Month | **public** **enum** WeekDay{  ***SUN***,***MON***,***TUE***,***WED***,***THU***,***FRY***,***SAT***  }  7 values are there under WeekDay |

* As we can see that (;) semicolon is optional
* The main objective of enum is to Define our own data type which is called enumerated data type
* Java introduced enum concept in java 1.5 V and it is more powerful when compared with old languages, because in old enum concept only named constant is allowed, while in java’s enum apart from named constant we can define several things like ( variable [int price], constructor, methods etc)

|  |
| --- |
| **package** com.enums.month;  **public** **enum** Month {  ***JAN***, ***FEB***, ***MAR***, ***APR***, ***MAY***, ***JUN***, ***JUL***, ***AUG***, ***SEP***, ***OCT***, ***NOV***, ***DEC***;  // Variable  **int** i;  // Inner Enum  **public** **enum** WeekDay {  ***SUN***, ***MON***, ***TUE***, ***WED***, ***THU***, ***FRY***, ***SAT***;  }  // Constructor  EnumMonth() {  }  // User defined method  **public** String display() {  **return** "Arun";  }  } |

**Internal Implementation of enum:**



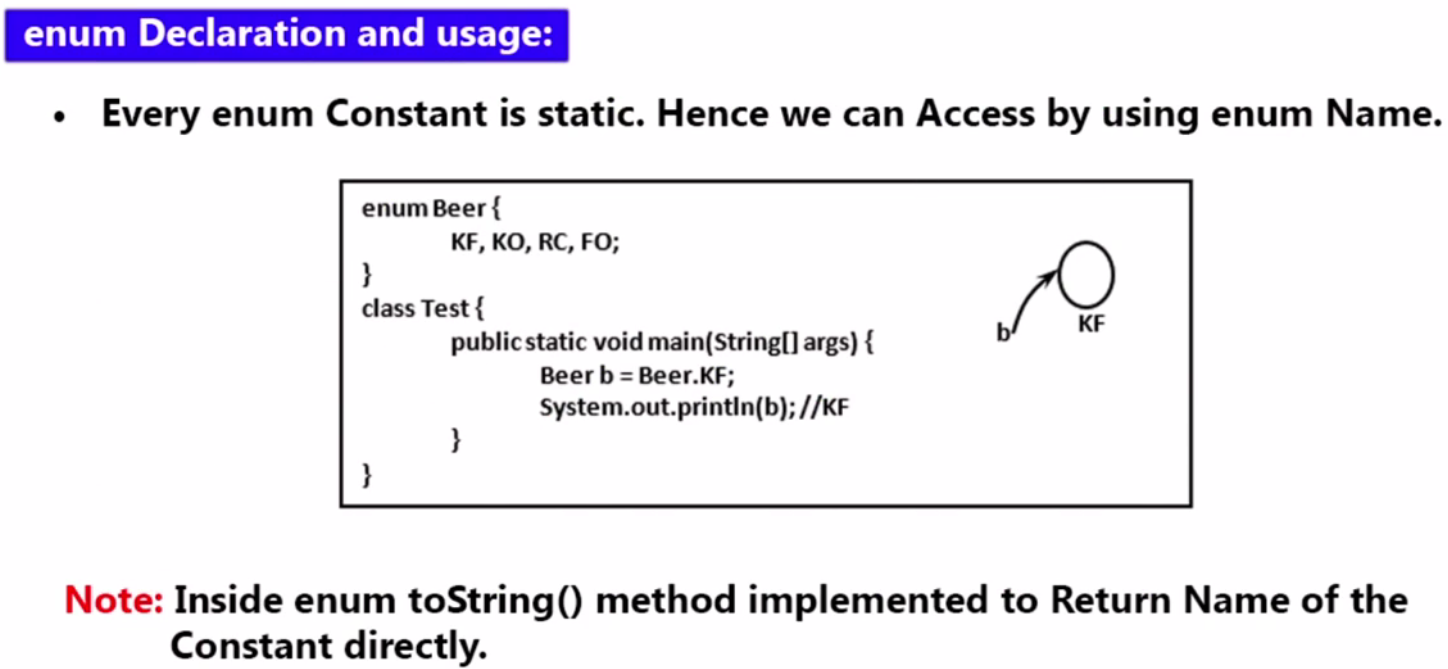
Every enum internally implemented by using class concept so internally **enum Beer {...}** gets converted into **Class Beer {….}.**

**Every enum constant is always public static final whether we define or not and every enum constant represents an object of the type enum, so when enum converts into class then the all the constant inside the enum will be converted into the object of that class as given above [public static final Beer KF = new Beer();]. So here we have**

|  |  |
| --- | --- |
| **public** **enum** WeekDay{  ***SUN***,***MON***,***TUE***;  } | **Class** WeekDay**{**  ***public static final Beer SUN = new WeekDay ();***  ***public static final Beer MUN = new WeekDay ();***  ***public static final Beer TUE = new WeekDay ();***  ***}*** |

**Enum and its equivalent class. So we can see that if enum constant is not there then how much lengthy code we have to write in java for named constant.**

**So wherever a predefined object is required then happily we can go for enum.**

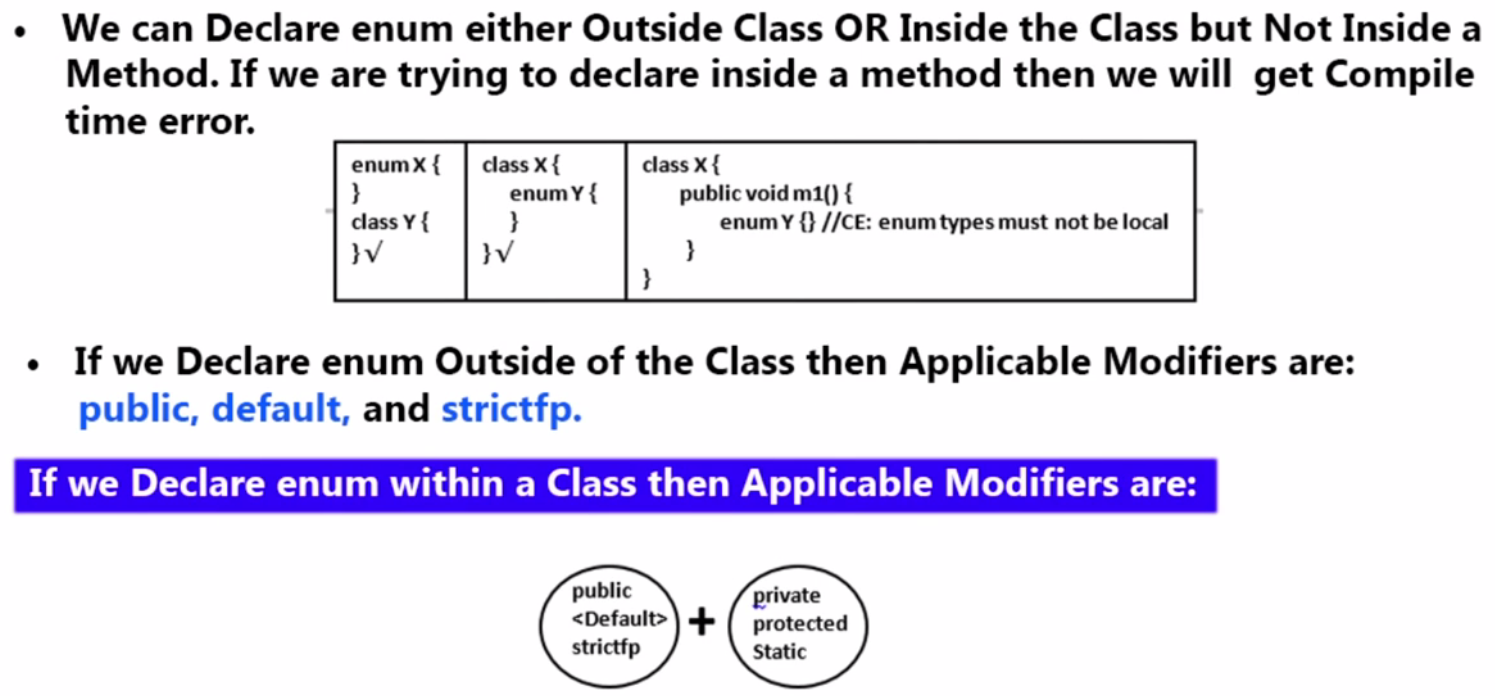
****

**Generally when we try to print any object reference then internally toString () method will be called. So here internally**

System.***out***.println (b) =🡺 will be = 🡺 System.***out***.println (**b.toString ()**);

|  |
| --- |
| **package** com.enums.month;  **import** com.enums.month.EnumMonth.WeekDay;  **public** **class** TestEnum {  **public** **static** **void** main(String[] args) {  EnumMonth j = EnumMonth.***JAN***;  EnumMonth.WeekDay sun = WeekDay.***SUN***;  System.***out***.println("Month Name-" + j + "-Week Day Name-" + sun); // Or  // System.***out***.println("Month Name-" + j. **toString ()** + "-Week Day Name-" + sun. **toString ()**);  }  }  **Output:**  Month Name-JAN -Week Day Name-SUN |

**Note**: Since the enum constant is always public static and final and hence it can be assessed with the **ClassName.enunConstant** that is why here we have accessed month and weekday enum values using like [**EnumMonth.*JAN*; & WeekDay.*SUN***]

****

**For enum declared outside the class, there are** **3 Modifiers:** [**public, default, final, abstract & strictfp]**

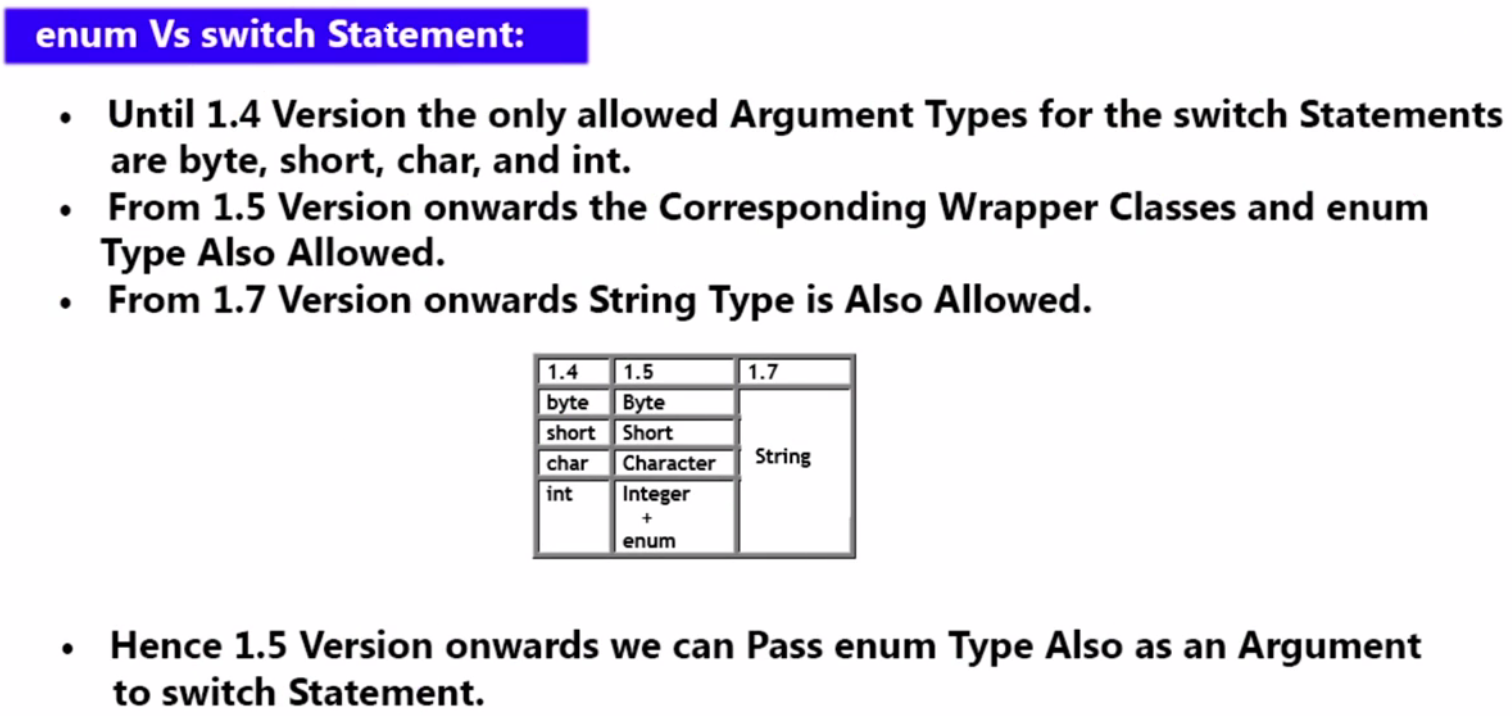
**For enum declared inside the class (or inner class), there are 6 Modifiers**: **[public, default, strictfp] + [private, protected, static]**

1. Every enum is always final implicitly but we can’t declare explicitly as final and that is why final is not allowed.
2. Abstract and final is illegal combination that is why abstract can’t be declared

|  |
| --- |
| **package** com.enums.month;  **import** com.enums.month.EnumMonth.WeekDay;  **public** **class** TestEnum {  **public** **enum** TimeZone {  ***Morning***, ***Evening***, ***Afternoon***, ***Midnight***;  }  **public** **static** **void** main(String[] args) {  EnumMonth j = EnumMonth.***JAN***;  //EnumMonth.WeekDay sun = WeekDay.SUN;  WeekDay sun = WeekDay.***SUN***;  TimeZone t = TimeZone.***Morning***;  System.***out***.println("Month Name-" + j + "Week Day Name-" + sun + "- at -" + t);  }  }  Output:  Month Name-JAN Week Day Name-SUN- at -Morning |

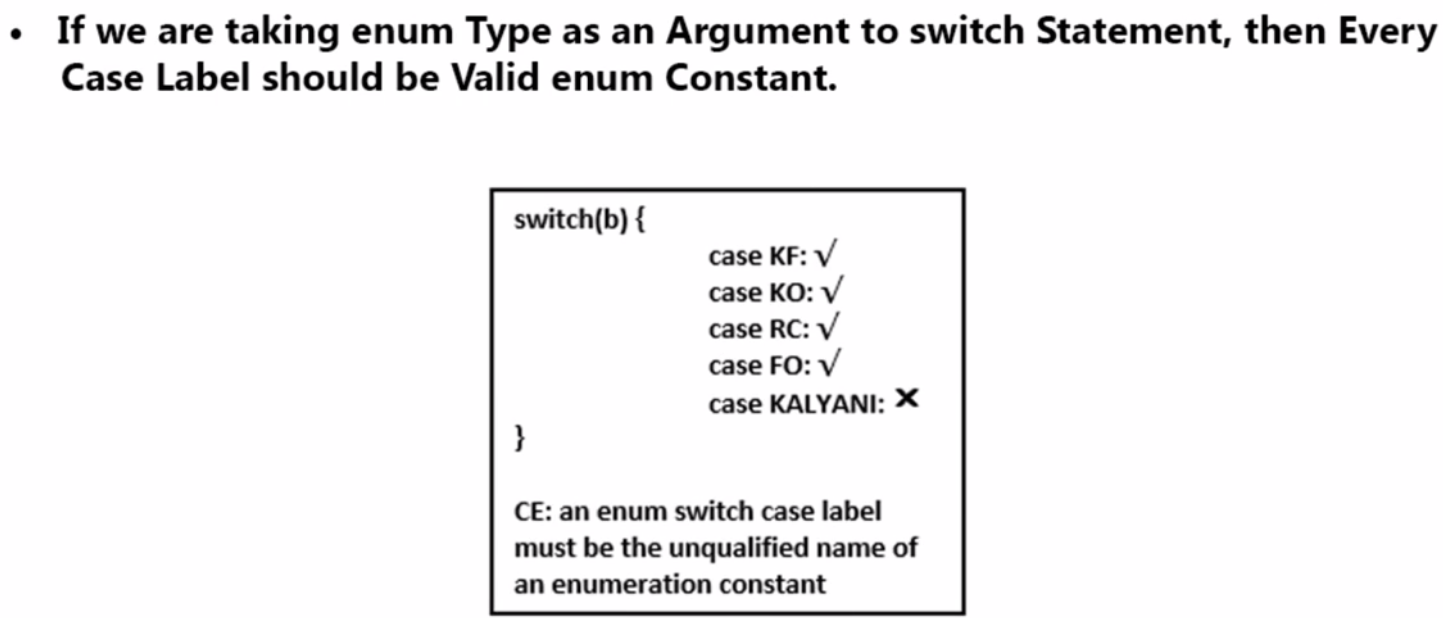
**Declared enum inside method:**

|  |  |
| --- | --- |
| **package** com.enums.month;  **import** com.enums.month.EnumMonth.WeekDay;  **public** **class** TestEnum {  **public** **static** **void** main(String[] args) {    //Declaed inside method  **public** **enum** TimeZone{  Morning,Evening,Afternoon,Midnight;  }  TimeZone t = TimeZone.Morning;  System.***out***.println("Time Zone -" + t);  }  } | Exception in thread "main" java.lang.Error: Unresolved compilation problems:  The member enum TimeZone can only be defined inside a top-level class or interface or in a static context |

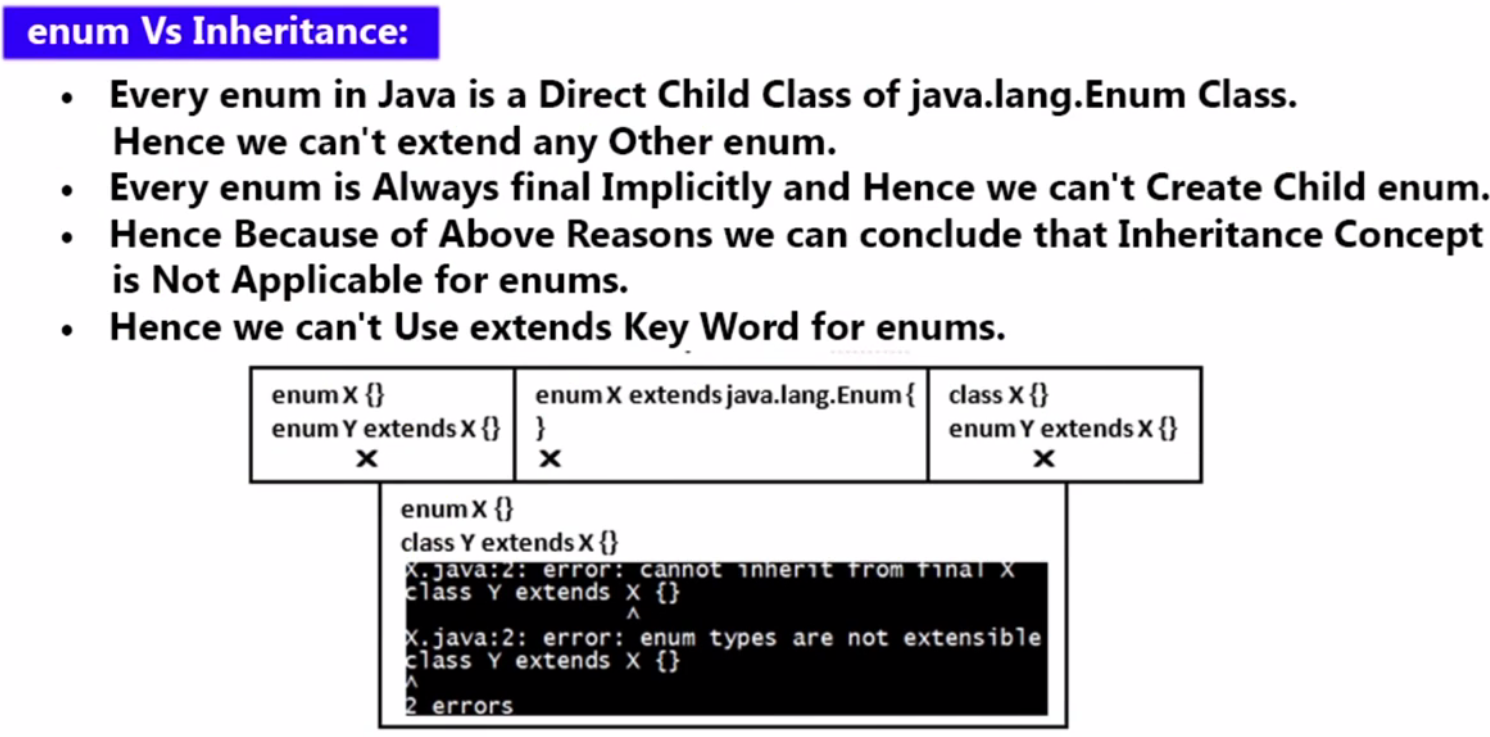
****

**Note: Apart from above if we take long, double, float, Boolean then will compile time error because they are not allowed inside the switch case:**

|  |  |
| --- | --- |
|  | **Allowed types in switch case:**  **Primitive Types(1.4):**  byte  short  char  int  **Wrapper types(1.5):**  Byte  Short  Character  Integer  **Enumeration Type(1.5):**  enum  **String Type(1.7):**  String |

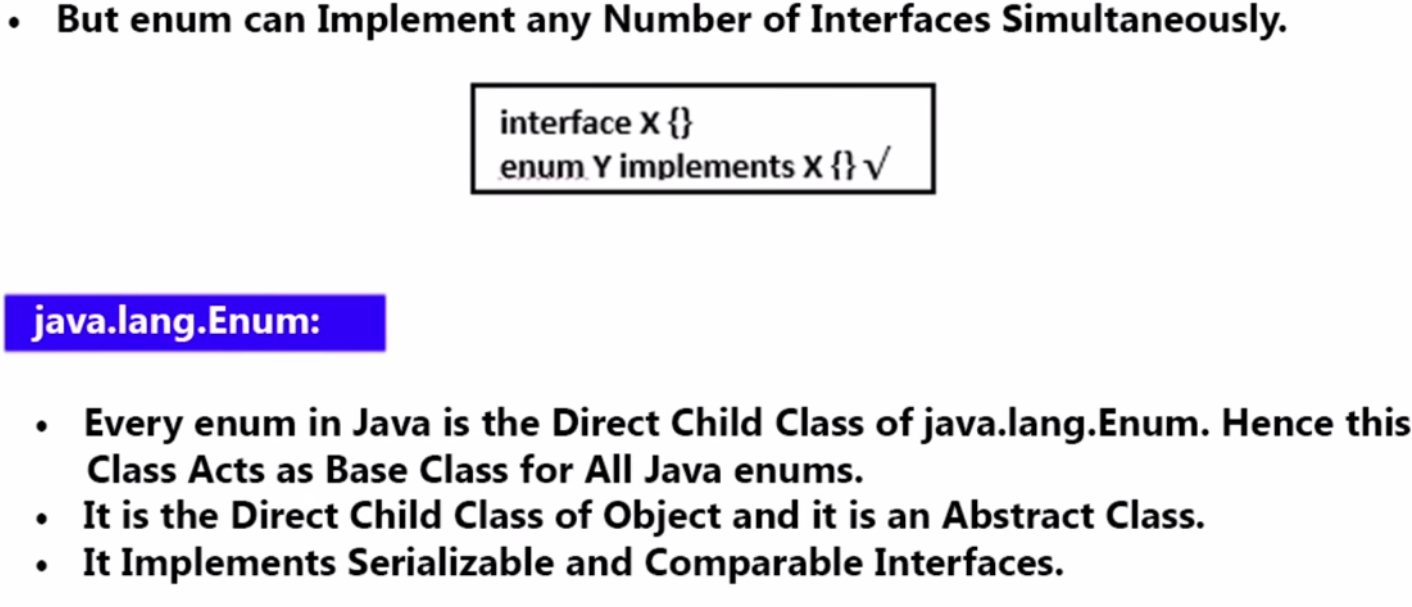
****

|  |  |
| --- | --- |
| **package** com.enums.month;  **public** **class** SwithcWithEnum {  **public** **enum** Week {  ***SUN***, ***MON***, ***TUE***, ***WED***, ***THU***, ***FRI***, ***SAT***;  }  **public** **static** **void** main(String[] args) {  Week w= Week.***MON***;  **switch**(w) {  **case** ***MON***:  System.***out***.println("Its Monday");  **break**;  **case** ***TUE***:  System.***out***.println("Its TUESDAY");  **break**;  **case** ***WED***:  System.***out***.println("Its Wednesday");  **break**;  **case** ***THU***:  System.***out***.println("Its Thursday");  **break**;  **case** ***FRI***:  System.***out***.println("Its Friday");  **break**;  **case** ***SAT***:  System.***out***.println("Its Saturday");  **break**;  **case** ***SUN***:  System.***out***.println("Its Sunday");  **break**;  //case OTH:  //System.out.println("Its OtherDay");  //break;  **default**:  System.***out***.println("No other day");  }  }  } | **Here OTH is not declare in the enum Week and we will get compile time error that is why I have comment it to avoid below compile time exception.**  Exception in thread "main" java.lang.Error: Unresolved compilation problem:  OTH cannot be resolved or is not a field  **Here w is MON so output will be:**  Its Monday |

****

**As we know that every class in java is the child class of Object class either directly or indirectly. But in case of enum there is no any indirect terminology i.e. every enum is the direct child of Enum class (java.lang.Enum class) and hence we cannot extend any other enum. So whatever enum (like enum Month, enum Week etc) we create is the direct child class of (java.lang.Enum) class. One more thing is every Enum is final implicitly so we cannot create child class also. So in short we can conclude that Inheritance concept is not applicable for enum**.

**In short extends keyword is not applicable for enum. But enum can implements any number of interfaces.**

****

|  |
| --- |
| **package** com.enums.month;  **public** **class** ExtendsConcept {    **enum** EnumMonth {  ***JAN***, ***FEB***, ***MAR***, ***APR***, ***MAY***, ***JUN***, ***JUL***, ***AUG***, ***SEP***, ***OCT***, ***NOV***, ***DEC***;  }  **enum** TimeZone {  ***Morning***, ***Evening***, ***Afternoon***, ***Midnight***;  }  **enum** WeekDay {  ***SUN***, ***MON***, ***TUE***, ***WED***, ***THU***, ***FRY***, ***SAT***;  }  **// Below extends is not applicable for enum**  **enum** Year **extends** EnumMonth{  ***YEAR1***, YEAR2  }  } |

**Java.lang.Enum- class:**

* It is the base class for all Java Enums
* It is an abstract class and we cannot create the object for this Enum class
* All the enum (including user-defined enums) is the direct child of Enum class
* It is the direct child class of Object class
* It implements Serializable and Comparable Interface so by default every user-defined enum implements Serializable and Comparable Interface, So we can compare two enum also

C:\Users\saraswati>javap java.lang.Enum

Compiled from "Enum.java"

**public abstract class java.lang.Enum<E extends java.lang.Enum<E>> implements java.lang.Comparable<E>, java.io.Serializable {**

public final java.lang.String name();

public final int ordinal();

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

public java.lang.String toString();

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

public final int hashCode();

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

public final int compareTo(E);

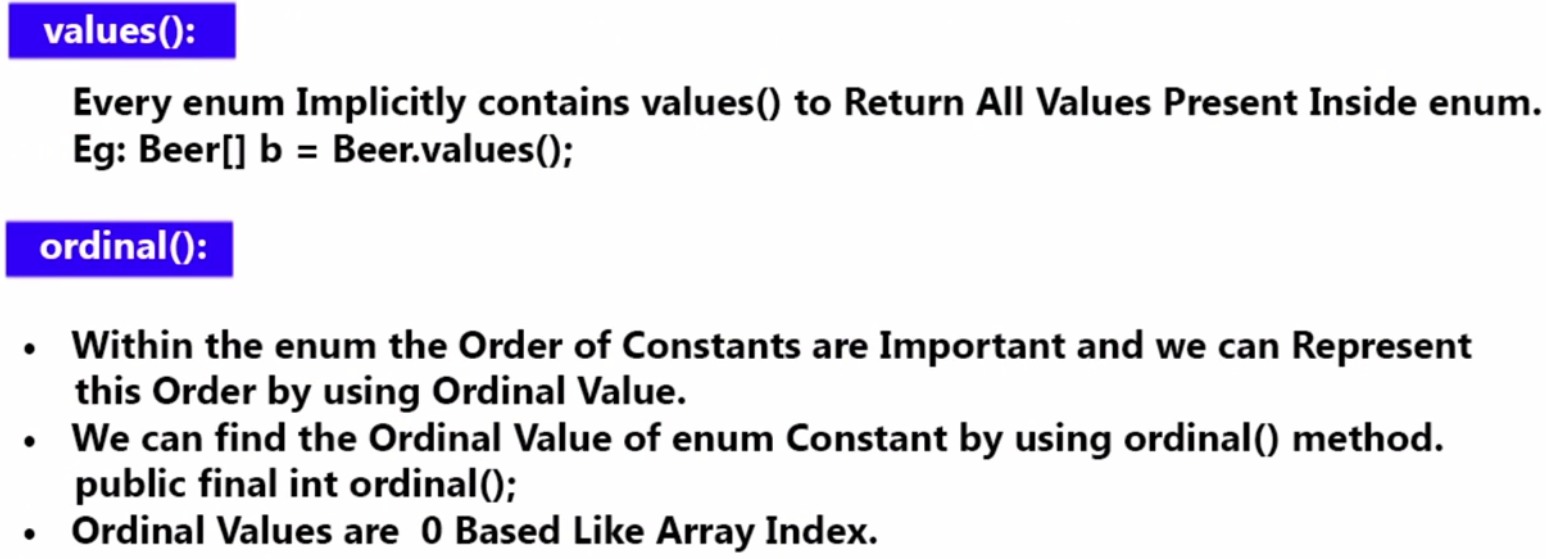
public final java.lang.Class<E> getDeclaringClass();

public static <T extends java.lang.Enum<T>> T valueOf(java.lang.Class<T>, java.lang.String);

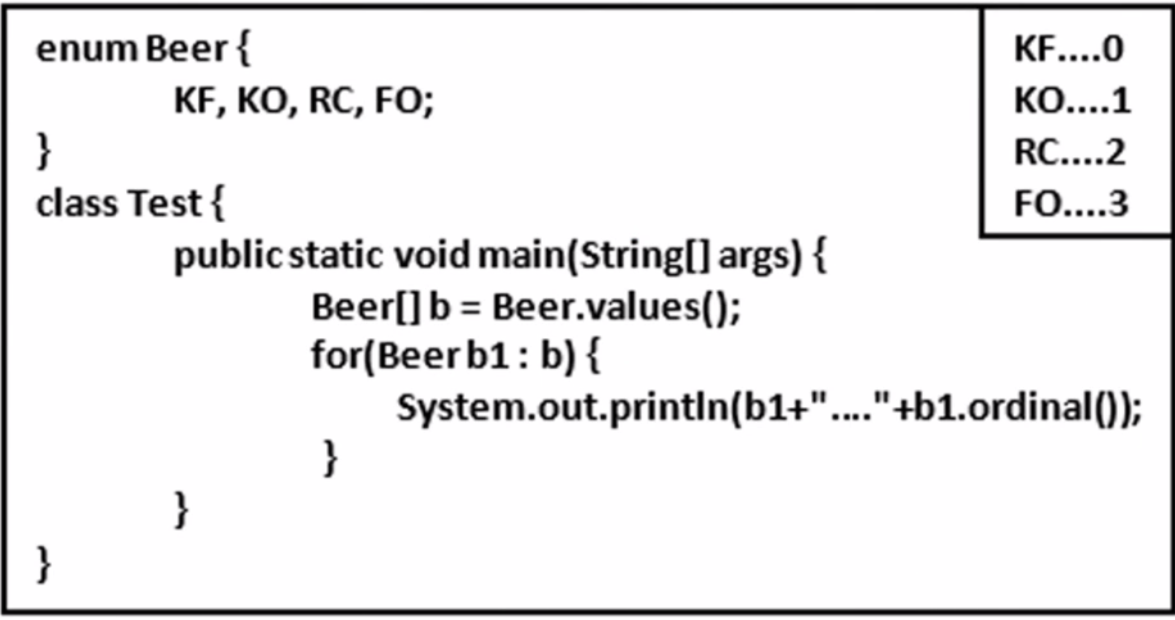
protected final void finalize();

public int compareTo(java.lang.Object);

}



|  |
| --- |
| **Note**-1: values () returns the constant value associated with the enum. Since it returns all the values as a group, that is why we have to take these enum values inside Arrays as declare in case of  Beer [] b =Beer.values (). So the objective of values is to list-out all the values present inside enum. |
| **Note2-:** This value () is not present anywhere in the Java API, Enum keyword itself provide this method implicitly that is why we don’t see this method () anywhere in the JAVA. |

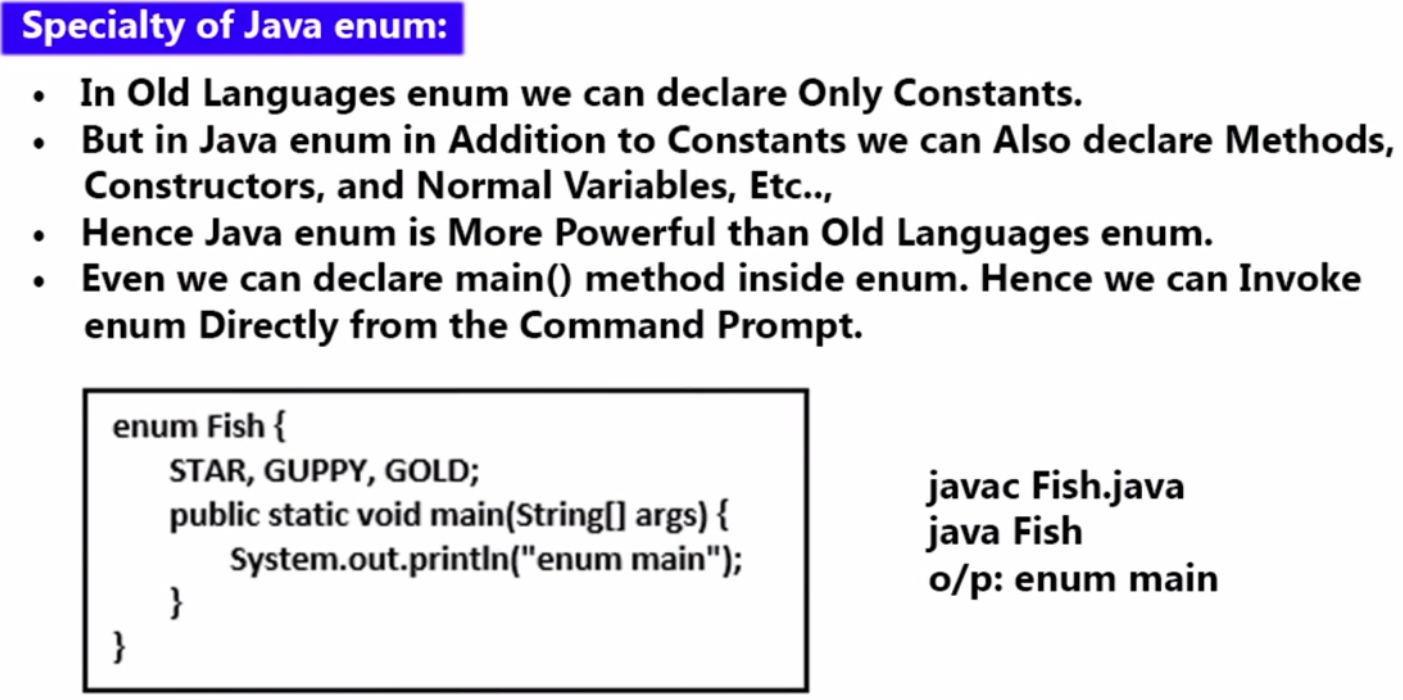


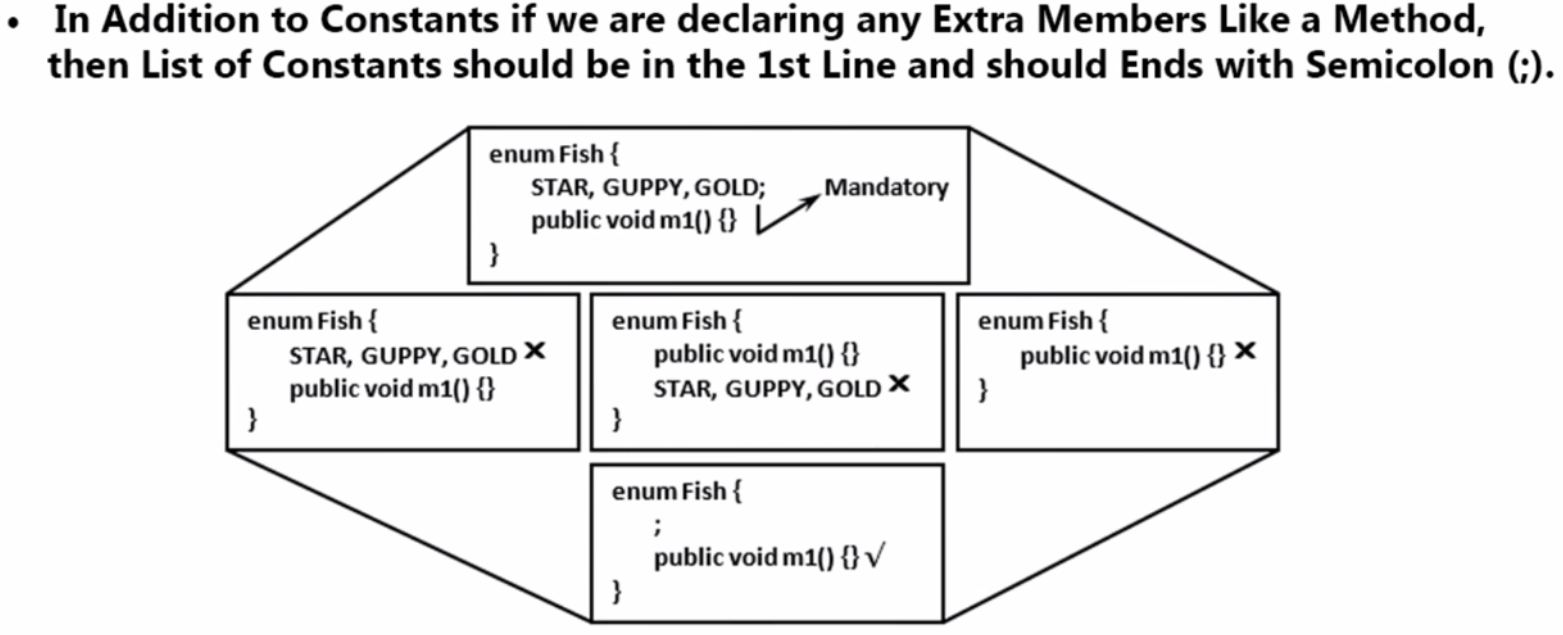
**int** [**java**](eclipse-javadoc:%E2%98%82=EnumConcept/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava)**.**[**lang**](eclipse-javadoc:%E2%98%82=EnumConcept/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang)**.**[**Enum**](eclipse-javadoc:%E2%98%82=EnumConcept/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(Enum.class%E2%98%83Enum)**.ordinal ():**

Returns the position in its enum declaration, where the initial constant is assigned an ordinal of zero). Most programmers will have no use for this method. It is designed for use by sophisticated enum-based data structures, such as [java.util.EnumSet](eclipse-javadoc:%E2%98%82=EnumConcept/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(Enum.class%E2%98%83Enum~ordinal%E2%98%82java.util.EnumSet) and [java.util.EnumMap](eclipse-javadoc:%E2%98%82=EnumConcept/C:%5C/Program%20Files%5C/Java%5C/jdk1.8.0_211%5C/jre%5C/lib%5C/rt.jar%3Cjava.lang(Enum.class%E2%98%83Enum~ordinal%E2%98%82java.util.EnumMap).

**Analogy**: As we know that in the Array index place very important role i.e. the index or order of Array element is very important, in the same way in enum also order of enum value is very important and can be find out by using ordinal() method of Enum class. So using the ordinal () method we can find out the order (index) of the enum value. It is very similar to Array in java, the only difference is that in Array the position or order of element is called index and here in case of enum it is called ordinal.

|  |  |
| --- | --- |
| **public** **enum** EnumMonth {  ***JAN***, ***FEB***, ***MAR, APR, MAY, JUN***;  }  **public** **static** **void** main(String[] args) {  EnumMonth[] m = EnumMonth.*values*();  **for** (EnumMonth m1 : m) {  System.***out***.println(m1 + "-at order-" + m1.ordinal());  }  } | JAN-at order-0  FEB-at order-1  MAR-at order-2  APR-at order-3  MAY-at order-4  JUN-at order-5 |





**Note:**



|  |  |
| --- | --- |
| **package** com.enums.month;  **public** **class** ExtendsConcept {  **enum** EnumMonth {  ***JAN***, ***FEB***, ***MAR***, ***APR***, ***MAY***, ***JUN***, ***JUL***, ***AUG***, ***SEP***, ***OCT***, ***NOV***, ***DEC***;  **int** i;  **public** **static** **void** main(String[] args) {  EnumMonth[] m = EnumMonth.*values*();  System.***out***.println(m);  }  **public** **void** display() {  //......  }  }  } |  |

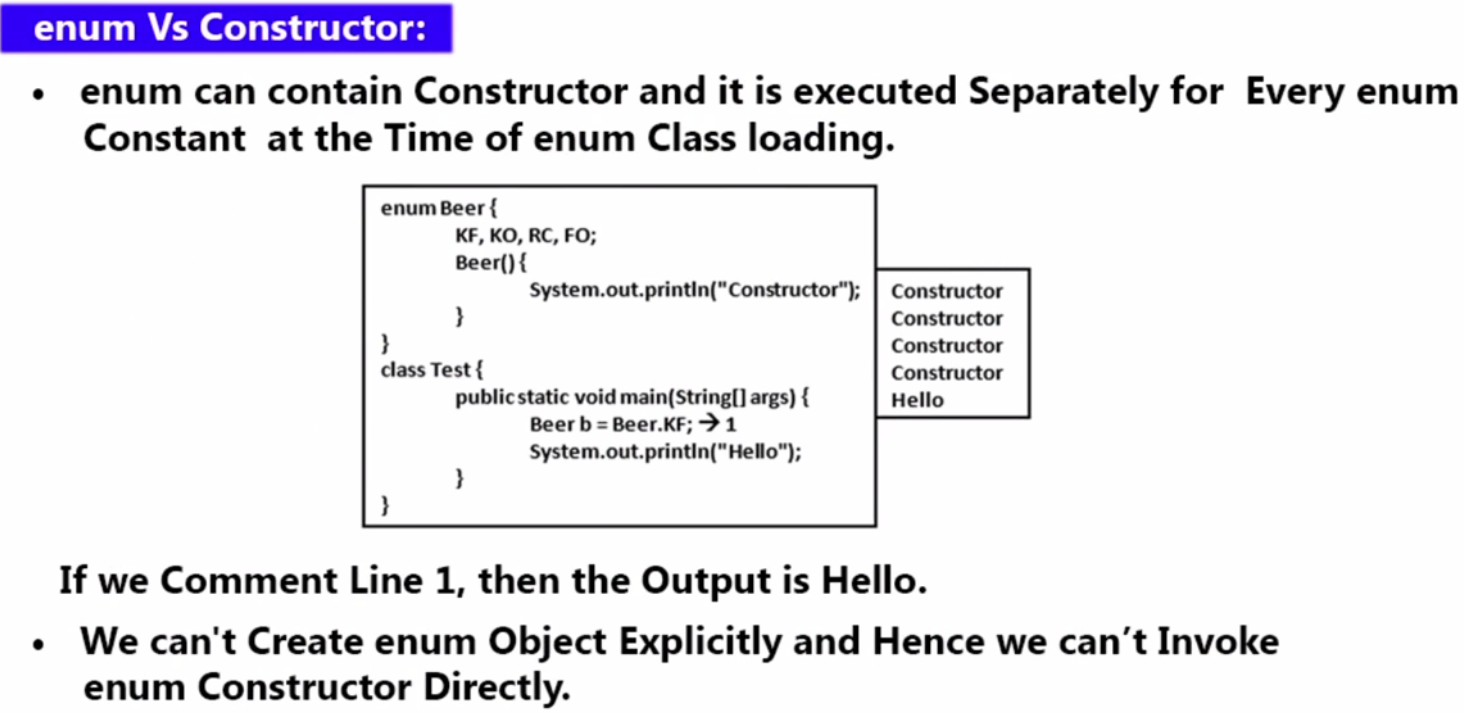
**Note:** As initially we learnt that (;) for constant value is optional but it is optional only, when have constant values only. But if along with the constant if we have any additional member like - variables, methods then compulsory we should put (;) after constant values otherwise we will get compile time error.

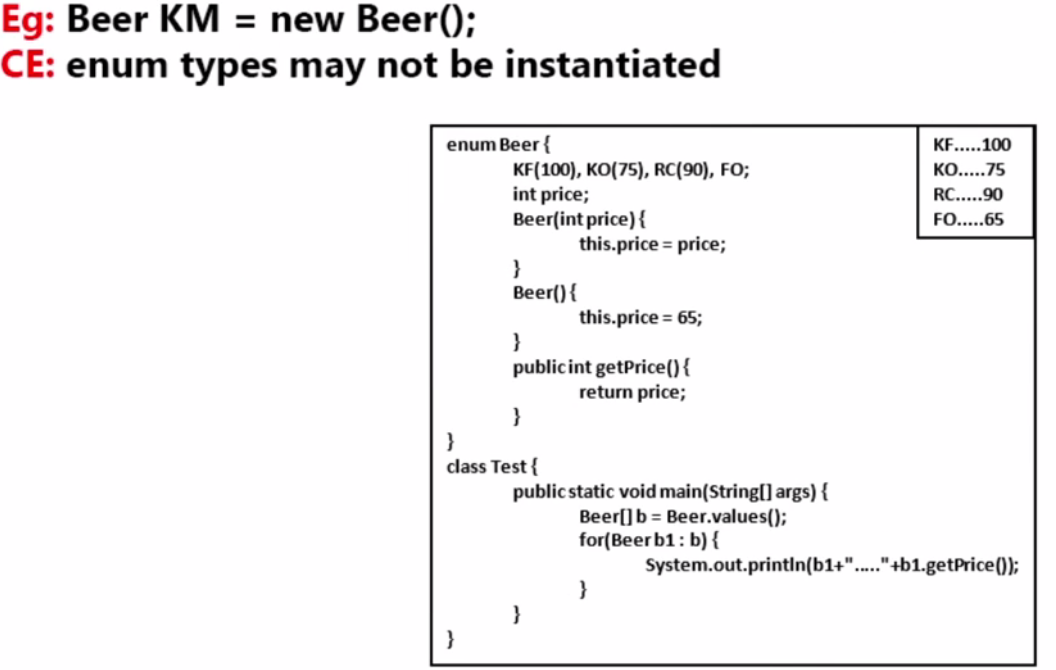
**Note:** Another note is when we have any additional member then at the first the constant values should be declared and end with (;) and after that other additional member should be declared otherwise we will get compile time error.

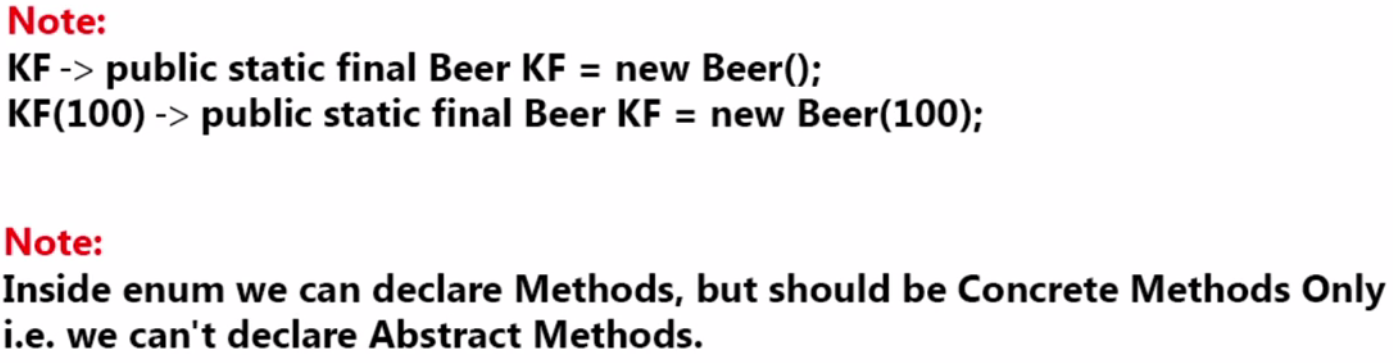
**Note:** If we are having only enum member without Constant values then we will get compile time error, because enum is specially for constant values and here we are not using the correct use of enum, and if want to have only other member apart from constant values then its better go with normal java class instead of using enum.

But if want to use enum only and we don’t want to have any constant values then at lease the first line of enum should have (;) to avoid compile time error. It means we have other member like – variable, methods, constructor without constant values but first line of the enum mandatory should have at least (;)

Note: We can declare enum without any constant values, and there is no use of empty enum without constant values.







As we know that in enum apart from constant values we can create the constructor of enum. Now we know that the constructor is called when the object is created. Then how it is possible to call enum constructor?

As we know that every constant value in the enum is the object itself of enum type. And every constant is public static and final so the object corresponding to these constant values will be created at the time of class load only and at the same time (class load time) enum constructor also will be called.

So in this way enum constructor gets called by its constant values only at the time of class loading.

In the first program we have created **enum Beer** inside class EnumConstrucotr and we have created main method also in the same class. Now if we run this program then two (.class) file will be created (**Beer.class & EnumConstrucotr.Class**) because enum Beer also a class of enum type. Now if run this program then only EnumConstrucotr class main method will be executed and we will get output: [Main Method called…]

|  |  |
| --- | --- |
| **package** com.enums.month;  **public** **class** EnumConstrucotr {  **enum** Beer {  ***KF***, ***KO***, ***RC***, ***RO***;  // enum constructor  Beer() {  System.***out***.println("Constructor called");  }  }  **public** **static** **void** main(String[] args) {    System.***out***.println("Main Method called..");  }  }  Output: Main Method called.. | **Reason**:  As we know that when we execute EnumConstrucotr java class then execution start from main () method and there is no any line which could create any link with enum Beer class and hence this Beer class will not be loaded and if Beer class will load then the object corresponding to enum class will not be created at the class load and thus enum constructor also will not be called.  This is why only main () method message will be displayed. |
| **package** com.enums.month;  **public** **class** EnumConstructor {  **enum** Beer {  ***KF***, ***KO***, ***RC***, ***RO***;  // enum constructor  Beer() {  System.***out***.println("Constructor called");  }  }  **public** **static** **void** main(String[] args) {  **Beer b = Beer.*KF*;**  System.***out***.println("Main Method called..");  }  }  **Output:**  Constructor called  Constructor called  Constructor called  Constructor called  Main Method called... | Reason: Here in this program we have got message from the enum constructor four times and main method message one time. The reason is, in the main method we have crated enum class object by accessing one constant value.  Now the question comes, here we have taken only one constant value from the enum Beer then constructor should be called only one time but here constructor got called four times?  A: So the reason is once we write [**Beer b = Beer.*KF*;]** then enum Beer class will be loaded into memory and as we already know that all he constant values of enum is public static and final and at the time loading only all the object corresponding to constant value will be created which in-turn call enum constructor and here we have four object will be created corresponding to four constant values and four time enum constructor will be called and hence we will get four time constructor message and one time main method message. |

So we can conclude that at the time of enum class loading all the constant values will be created and all the constant values will be converted into object and which in-turn calls enum constructor. And number of constructor calling will be equal to number of constant present in the enum class.

Q: Why we are creating enum Beer object by taking enum constant values like [**Beer b = Beer.*KF*;]** instead of Beer b = new Beer ();]

A: Since enum is basically used for constant values and if we create the object of enum Beer class using new operator then it will be like normal class which violate enum concept and we will get compile time error. If we want to create object using new operator then instead of using enum we can go for normal class why to take enum.

So we cannot create enum constructor explicitly by using new operator; and if we try to create enum class object using new operator then we will get compile time exception saying [Cannot instantiate the type EnumConstructor.Beer]

So from the above discussion we have got to know, how to call default constructor.

Q: How to call parameterized constructor?

A: As we know that, during the enum class loading all the enum constant gets converted into corresponding equivalent object as given below.

|  |  |
| --- | --- |
| Default constructor | Parameterized Constructor |
| KF 🡺Public static final Beer KF = new Beer() | KF(100)🡺Public static final Beer KF = new Beer(100) |
| KO🡺Public static final Beer KO = new Beer() | KO(75) 🡺Public static final Beer KO = new Beer(75) |
| RC🡺Public static final Beer RC = new Beer() | RC(90) 🡺Public static final Beer RC = new Beer(90) |
| RO🡺Public static final Beer RO = new Beer() | RO(65)🡺Public static final Beer RO = new Beer(65) |

So if we want to invoke parameterized enum constructor then we have to mentions the enum constant values as give above in the RHS table.

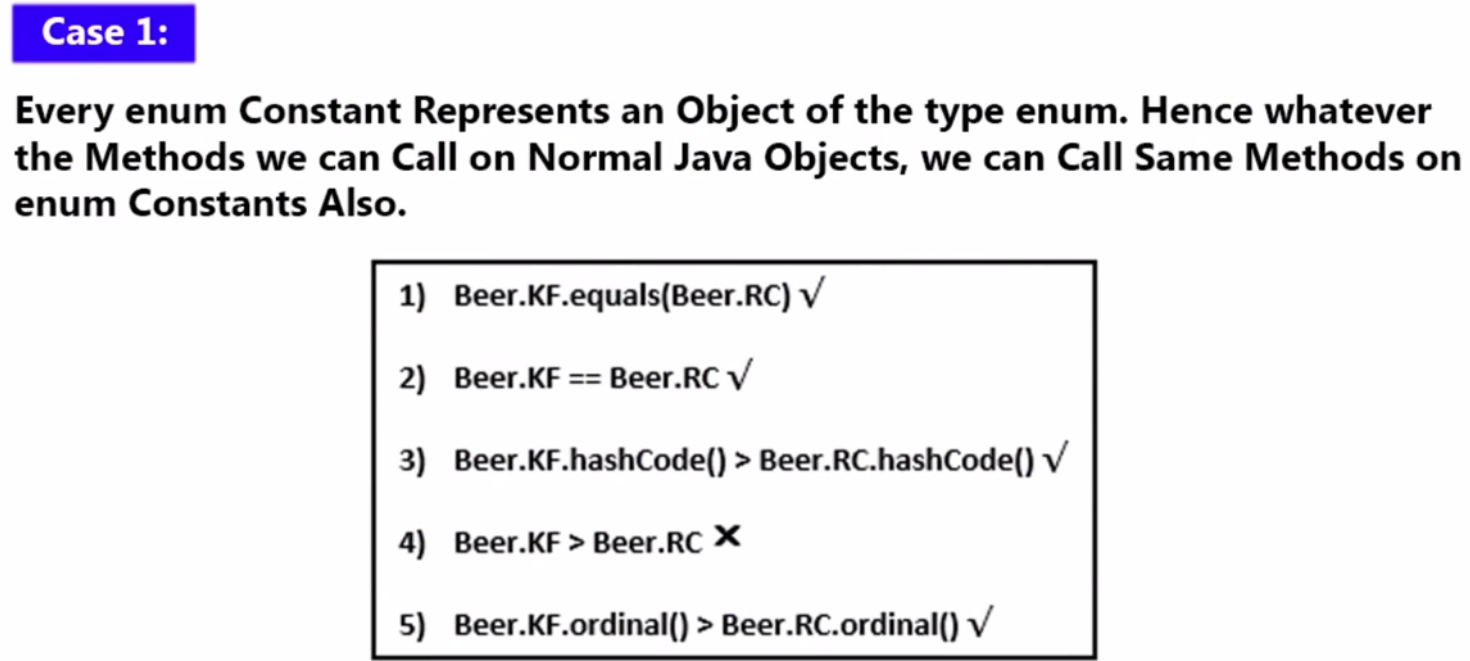
|  |  |
| --- | --- |
| **package** com.enums.month;  **public** **class** EnumConstructor {  **enum** Beer {  ***KF***(100), ***KO***(90), ***RC***(75), ***RO***;  **int** price;  Beer() {  System.***out***.println("Default Constructor called");  **this**.price = 65;  }  Beer(**int** price) {  **this**.price = price;  System.***out***.println("Parameterized Constructor called");  }  **public** **int** getGetPrise() {  System.***out***.println("getter called");  **return** price;  }  }  **public** **static** **void** main(String[] args) {  System.***out***.println("Main Method called..");  Beer[] b = Beer.*values*();  **for** (Beer b1 : b) {  System.***out***.println(b1 + "-" + b1.ordinal() + "-" + b1.getGetPrise());  }  }  } | Output:  Main Method called..  Parameterized Constructor called  Parameterized Constructor called  Parameterized Constructor called  Default Constructor called  getter called  KF-0-100  getter called  KO-1-90  getter called  RC-2-75  getter called  RO-3-65 |
| 1. Created three constant with prize and one constant without prize 2. Created one default and one parameterized constructor 3. So three constant with values will call parameterized constructor three time and initialize the prize 4. Default constructor will be called one time where we have set default prize 5. In the prize getter method we have returned the prize initiated by constructor 6. In the main method collect all the group of constant into array Beer[] b and then print Beer name and prize using for each loop | |

So here in the above example along with constant we have following things

1. Variable [int prize]
2. Default constructor
3. Parameterized constructor
4. Method getPrize() :

Note: The method declared inside enum should be concrete method only. We cannot declare abstract method inside enum because

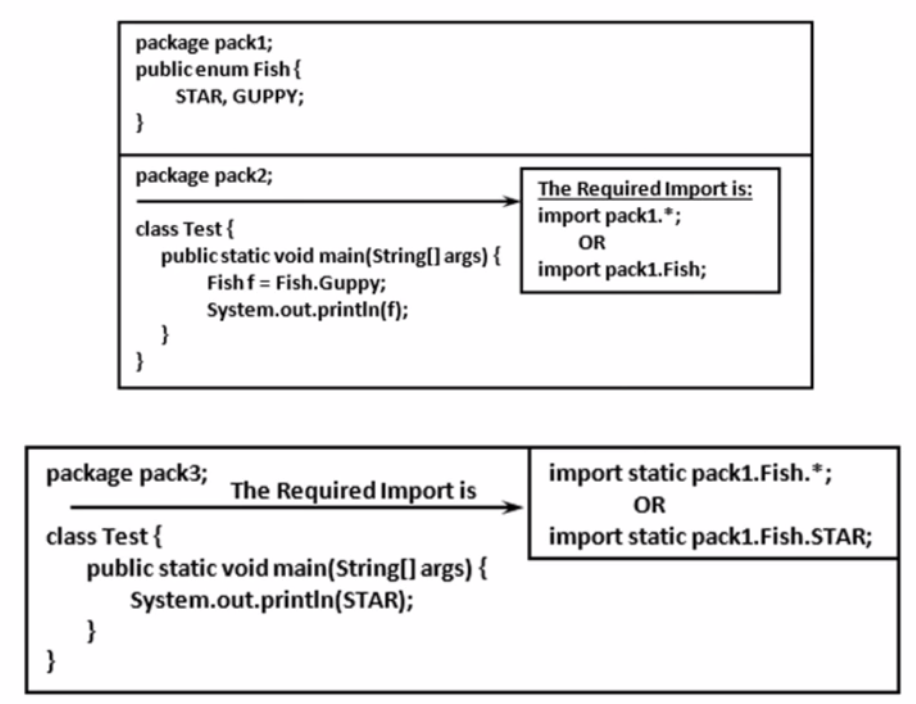
1. If we declare any abstract method inside the class then class also must be abstract and since enum class is always final and we cannot declare [final abstract] simultaneously otherwise we get illegal combination.
2. Another reason is if we have abstract method inside that class then compulsory those methods should be implemented in its child class and since enum is already final so it is not possible to created child of enum class.



1. The first one is correct because Beer.KF and Beer.RC represent object of enum types and to object can be compared using Object’s equal() method
2. To objects can be compared using (==) equal operator also so second one is also correct
3. Let’s see first 4th one, here two enum object trying to compare using operational operator and operational operator is not allowed for objects for complarision because an object refers just an entity not the entity value and there is no benefit comparing two object.
4. In the third one, we are getting the hashCode of the object (which is nothing but integer number) so third one is correct because here we are trying to compare two object’s hashCode
5. In the last one we are trying to compare the ordinal (index of enum constant value) so this is correct.



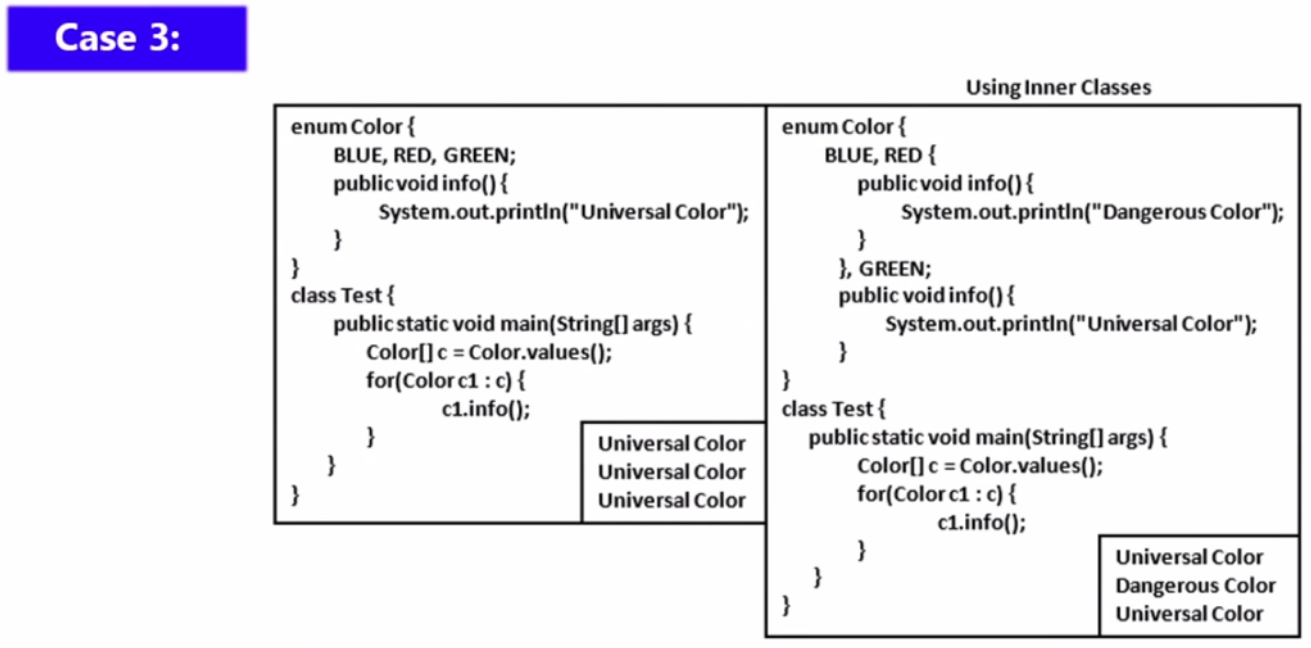
Here in the above program we have done normal import for ArrayList class and static import for sqrt (4) and in that case we don’t need to use className.staticMethod() to access static method of the class.



Here in case of Enum class is static class and it constant value is also static so if we want to access constant value then we have use ClassName.contantValue like we have done for GUPPY [ Fish f = Fish.Guppy] .

But if we want to access constant value without using ClassName then in that case we have to go with static import like [import static pack1.Fish.STAR] or [import static pack1.Fish.\*] and now we can access static variable or static method without using ClassName.

|  |  |
| --- | --- |
|  | Here  Fish f = Fish.STAR;  Here normal import will be required like  Import pack1.Fish or import pack1.\*  But for the below statement static import is required because here we are trying t access static constant value without using ClassName so in this case we have give full qualified name of the static constant with static import  System.out.println(GUPPY)  Import static pack1.Fish.GUPPY  Or  Import static pack1.Fish.\* |



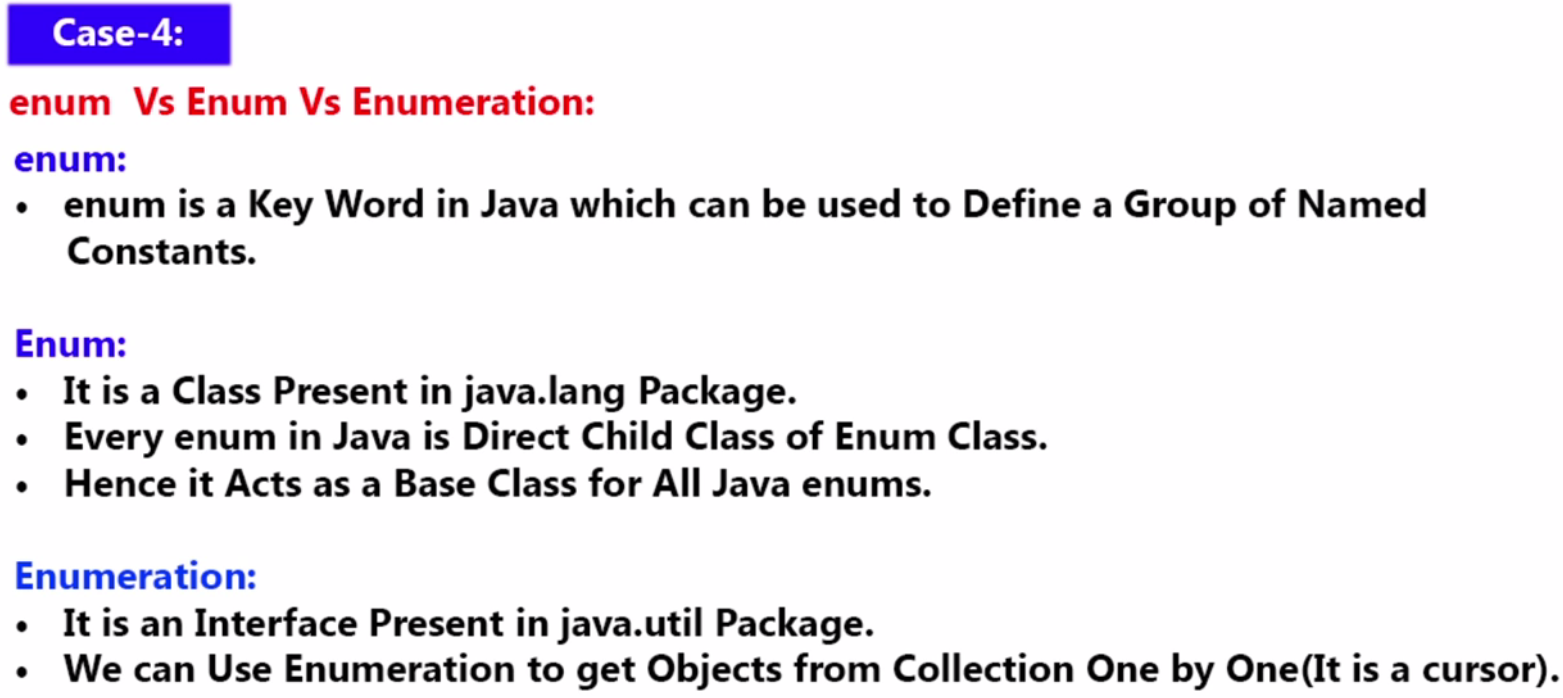
1. Here we have num Color and created three color constant BLUE, RED, GREEN
2. In the first program we just simply created on concrete method to display info of color
3. In the main method we have collected all the color into Array c and then using for each loop we calling concrete method info()

In the second program while create color as constant value then we have created an anonymous inner class for color RED constant and for RED color we are overriding info() method

So for BLUE – Universal color

For Red -- Dangerous Color

For Green – Universal Color



**Enum is base class for enum class keyword and Enumeration is an Interface**