Enums

- Enums are introduced in jdk1.5
- Enums are **Fixed no. of, Similiar kind of Constants.**
- Because they are constants, the names of an enum type's fields are in uppercase letters.
- Every enum constant is **public, static & final** by default.
- Every enum constant can be represented as a valid java identifier.
- Every enum constant should be **unique.**
- All enum constants should be **delimeted with comma (,)**
- Inside enum, not only enum constats are there,
  we can keep **attributes, methods, constructors, IIB's, SIB's.**
- enum can be meber of .java file.
- enum can be member of a class.
- For enums also .class files are generated.
- Among all members of one .java file, only one member can be public.
  Whichever member is public, then that name only can be used for saving the .java file.

  If enum is public, then that .java file should be saved as enum name.
- enum improves type safety
- enum can be easily used in switch
- enum may implement many interfaces but cannot extend any class because it internally extends Enum class

Program 1:

```
enum A {

}
```

Explanation:

```
enum A is a member of .java file.
For enum also .class file is generated.
Here A.class file is generated.

enum itself static by default.
```

enum 'A' extends java.lang.Enum.

java.lang.Enum is an abstract class.

This is the common base class of all Java language enumeration types.

It is declared as follows:

public abstract class Enum extends Object implements Comparable, Serializable

This clearly means that enums are **comparable and serializable implicitly**.

All enum types in java are **singletonby default.**
So, you can **compare enum types using '==' operator also**.

This also means that all enums extends java.lang.Enum, so they **can not extend any other class** because java does not support multiple inheritance this way. But, **enums can implement any number of interfaces**.

Program 2:

```
public enum B {

}

class C {

}

interface D {

}

@interface E1 {

}
```

Explanation:

enums, classes, interfaces and annotations can become member of .java file.
Here totally 4 .class files are generated.

Examples of enum that is defined outside the class:

Program 3:

```
enum E {

        C1, C2, C3, C4;
}

class F {
        public static void main(String[] args) {

                System.out.println("E.C1 : " + E.C1);
                System.out.println("E.C2 : " + E.C2);
        }
}
```

Output:

```
E.C1 : C1
E.C2 : C2
```

Explanation:

enum E contains 4 constants. All are unique.

It is advisable to keep all the constants in Uppercase.

All enum constants are public & static by default.

enum constants are static, Becoz of that only, we're accessing the enum constants by using class_name.

Program 4:

```
enum G {

        CON1, CON2, CON3, CON4
}

class H {
        public static void main(String[] args) {

                G g1 = G.CON1;
                G g2 = G.CON2;

                System.out.println("g1 : " + g1);
                System.out.println("g2.toString() : " + g2.toString());
        }
}
```

Output:

```
g1 : CON1
g2.toString() : CON2
```

Explanation:

enum G has 4 constants.

Here ';' is optional at the end of enum constant. Becoz enum 'G' having only constants & its not having any other members (like constructors or methods).

In main(),
g1 is a 'G' type refrence which is a G type and it can be initialized with 'G' types constant.

Here 'g1' is which is pointing 'CON1' constant.

g2 is a G type refrence which is pointing 'CON2' constant.

toString() method got overrided in enum. That is why if we print enum refrence, we're getting enum constant.

Program 5:

```
enum I {
	TEST1, TEST2, TEST3, TEST4
}
class J {
	public static void main(String[] args) {

		I t1 = I.TEST1;
		System.out.println("t1 : " + t1);
		t1 = I.TEST2;
		System.out.println("t1.toString() : " + t1.toString());
	}
}
```

Output:

```
t1 : TEST1
t1.toString() : TEST2
```

Program 6:

```
enum K {

	T1, T2, T3, T4
}

class L {
	public static void main(String[] args) {

		K k1 = K.T1;
		System.out.println("k1 : " + k1);

		k1 = K.T2;
		System.out.println("k1 : " + k1);

		k1 = K.T3;
		System.out.println("k1 : " + k1);

		k1 = K.T4;
		System.out.println("k1 : " + k1);
	}
}
```

Output:

```
k1 : T1
k1 : T2
k1 : T3
k1 : T4
```

Program 7:

```
enum Month {

	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEPT, OCT, NOV, DEC
}

class U {
	public static void main(String[] args) {

		Month m3 = Month.MAR;
		System.out.println("m3 : " + m3);

		Month m5 = Month.MAY;
		System.out.println("m5 : " + m5);

		System.out.println("7th month : " + Month.JUL);
	}
}
```

Output:

```
m3 : MAR
m5 : MAY
7th month : JUL
```

Explanation:

Enum is a fixed number of, similiar kind of Constatns.

Here all the months are fixed & all are similiar kind.

That is why these constatns are represented inside enum.

Program 8:

```
public class M1 {

	enum En1 {

		C1, C2, C3, C4;
	}
}

class M2 {

	public static void main(String[] args) {

		M1.En1 e1 = M1.En1.C1;
		M1.En1 e4 = M1.En1.C4;

		System.out.println("e1 : " + e1);
		System.out.println("e4 : " + e4);

		e4 = M1.En1.C3;

		System.out.println("e4.toString() : " + e4.toString());
	}
}
```

Output:

```
        e1 : C1
        e4 : C4
        e4.toString() : C3
```

Explanation:

This .java file contains 2 members, one enum and one class.

enums can become member of a class.

Here enum 'En' is a member of class 'M'.

Eventough enum is a member of a class, For this enum 'En1' also .class(M1$En1.class) file is generated.

To access 'En1' enum in outside of 'M1' class, it should be accessed along with outer_classname.

Here, M1.En1

C1 can be accessed outside like M1.En1.C1

Example of enum that is defined within the class:

Program 9:

```
        public class M {

                enum En {

                        C1, C2, C3, C4;
                }

                public static void main(String[] args) {

                        En e1 = En.C1;
                        En e2 = En.C2;

                        System.out.println("e1 : " + e1);
                        System.out.println("e2 : " + e2);

                        e2 = En.C4;

                        System.out.println("e2.toString() : " + e2.toString());
                }
        }
```

Output:

```
        e1 : C1
        e2 : C2
        e2.toString() : C4
```

Explanation:

enums can become member of a class.

Here enum 'En' is a member of class 'M'.

Eventough enum is a member of a class, For this enum 'En' also .class(M$En.class) file is generated.

class 'M' contains 2 members. one is a enum, another one is a main() method.

Example of enum with constructors:

Note : Constructor of enum type is private if you don't declare private compiler internally have private constructor

Program 10:

```
	enum Season{

		WINTER(10),SUMMER(20);

		private int value;

		Season(int value){
			this.value=value;
		}
	}
```

Internal code generated by the compiler for the above example of enum type

```
	final class Season extends Enum
	{
		private Season(String s, int i, int j)
		{
			super(s, i);
			value = j;
		}

		public static final Season WINTER;
		public static final Season SUMMER;
		private int value;

		static
		{
			WINTER = new Season("WINTER", 0, 10);
			SUMMER = new Season("SUMMER", 1, 20);
		};
	}
```

Program 11:

```
enum V {

	C1, C2, C3;

	V() {
		System.out.println("enum constructor V()");
	}
}

public class W {
	public static void main(String[] args) {

		System.out.println("Main Begin");

		V v3 = V.C3;
		System.out.println("v3 : " + v3);
		V v2 = V.C2;
		System.out.println("v2 : " + v2);

		System.out.println("V.C2 : " + V.C2);
	}
}
```

Output:

```
Main Begin

enum constructor V()
enum constructor V()
enum constructor V()

v3     : C3
v2     : C2
V.C2   : C2
```

Explanation:

enum can contains constructors also.

Inside enum, we can keep contructors, methods, attributes, SIB's & IIB's.

Here, enum constatns should end with semicolon (;).
Becoz along with enum constants, constructors also there.

If only enum constants are presents in 'enum' means, semicolon is optional.
If some other members also inside means, semicolon is mandatory.

Whenever 'enum' is loading into memory, at that time each constant is loading into memory, at that time, corresponding constructor also executing.

Becoz enum constants are static, so all enum constants are loading into memory whenever particular 'enum' is loading into memory.

Here,

whenever enum 'V' is loading into memory, all enum constants also loading into memory one by one.

1st constant 'C1' is loading into memory, that time no arg constructor is executing automatically.
then constant 'C2' is loading into memory, that time no arg constructor is executing automatically.
then constant 'C3' is loading into memory, that time no arg constructor is executing automatically.

So, totally, C() constructor is executed 3 times, becoz 3 constants are loaded into memory.

Once all enum constants are loaded in memory, that enum constants are available in the memory, later we can use that constants in our program.

Program 12:

```
public class X {

	enum A {		// static by default. (i.e) static enum A

		CON1(100), TEST(200);

		A(int i) {
			System.out.println("A(int i) : " +i);
		}
	}

	public static void main(String[] args) {
		A a1 = A.CON1;
		A a2 = A.TEST;

		System.out.println(a1);
		System.out.println(a1);
	}
}
```

Output:

```
A(int i) : 100
A(int i) : 200

CON1
CON1
```

Explanation:

whenever class 'X' is loading into memory, all static members are loading into memory. enum itself static by default.

Here class 'X' contains 2 static members, one is enum 'A', 2nd one is main() method. So both static members are loading into memory one by one.

Whenever enum 'A' is loading into memory. all enum constants are loading into memory one by one.

while loading each constant one by one, automatically correspoding constructors also executed automatically.

Here,

1st 'CON1' constant is loading into memory which consists '100' as argument, so single argumented constructor is executing.
then 'TEST' constant is also loading into memory which consists '200' as argument, so single argumented constructor is executing.

So, totally, A(int i) constructor is executed 2 times, becoz 2 constants are loaded into memory.

Once all enum constants are loaded in memory, that enum constants are available in the memory, later we can use that constants in our program.

Program 13:

```
public class Y {

	enum A {

		CON1, CON2(100), CON3(50.5f), CON4("SASHI");

		A() {
			System.out.println("A()");
		}
		A(int i) {
			System.out.println("A(int) : " +i);
		}

		A(float f) {
			System.out.println("A(float) : " +f);
		}

		A(String s) {
			System.out.println("A(String) : " +s);
		}
	}

	public static void main(String[] args) {

		A c1 = A.CON1;
		A c3 = A.CON2;

		System.out.println("c1 : " +c1);
		System.out.println("c3 : " +c3);
		System.out.println("A.CON4 : " + A.CON4);
	}
}
```

Output:

```
A()
A(int) : 100
A(float) : 50.5
A(String) : SASHI

c1 : CON1
c3 : CON2
A.CON4 : CON4
```

Explanation:

enum 'A' contains totally 4 types of constants.

For every constant, there should be a corresponding constructor, then only compilation success.

If any constructor is missing, then compile time error.

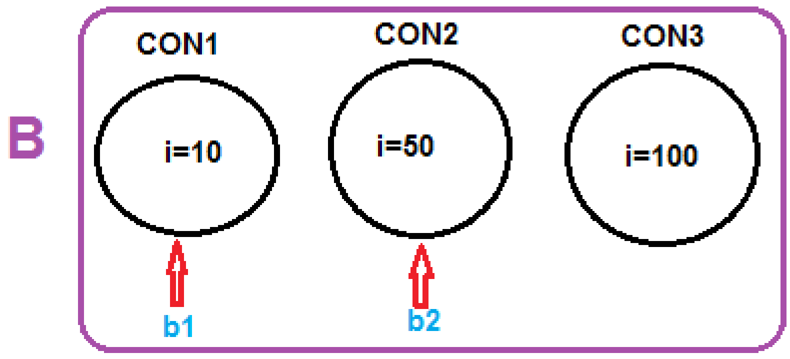
Program 14:

```
	public class Z {
		public enum B {
			CON1(10), CON2(50), CON3(100);
			int i;
			B(int i) {
				System.out.println("B(int)");
				this.i = i;
			}
		}

		public static void main(String[] args) {
			B b1 = B.CON1;
			B b2 = B.CON2;

			System.out.println("b1 : " + b1);
			System.out.println("b2 : " + b2);

			System.out.println("b1.i : " + b1.i);
			System.out.println("b2.i : " + b2.i);

			System.out.println("----------------------------------------------");
			System.out.println("Using values() method with Enhanced for loop");
			System.out.println("----------------------------------------------");

			for (B b : B.values()) {
				System.out.println("b.i : " + b.i);
			}
		}
	}
```

Output:

```
	B(int)
	B(int)
	B(int)

	b1 : CON1
	b2 : CON2
	b1.i : 10
	b2.i : 50

	----------------------------------------------
	Using values() method with Enhanced for loop
	----------------------------------------------

	b.i : 10
	b.i : 50
	b.i : 100
```

Diagram:



Explanation:

Inside enum 'B', 3 constants & one attribute 'i' is available.

For each enum constant, one copy of attribute 'i' is available.
Each enum constant contains one 'i'attribute.

Whenever enum constants are loading into memory, constructors are executed.
Along with that all non static members (attributes & methods) are loading into memory.

while loading 'CON1', 'i' is loading into memory and becoming part of 'CON1'
while loading 'CON2', 'i' is loading into memory and becoming part of 'CON2'
while loading 'CON3', 'i' is loading into memory and becoming part of 'CON3'

In main() method,
'b1' is pointing 'CON1', inside 'CON1' there is 'i' attribute, that 'i' value is '10'.
'b2' is pointing 'CON2', inside 'CON2' there is 'i' attribute, that 'i' value is '50'.

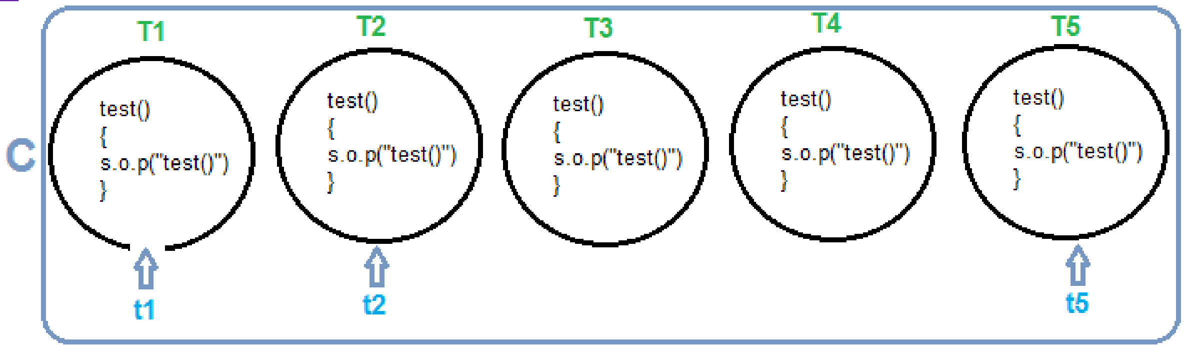
Program 15:

```
public class Z1 {

	enum C {

		T1, T2, T3, T4, T5;

		void test() {
			System.out.println("test()");
		}
	}

	public static void main(String[] args) {

		C t1 = C.T1;
		C t2 = C.T2;
		C t5 = C.T5;

		t1.test();
		t2.test();
		t5.test();

		C.T4.test();
	}
}
```

Output:

```
test()
test()
test()
test()
```

Diagram:



Explanation:

Inside enum 'C', 5 constants & 1 method is there.

For each constant, one copy of test() method is there.

Whenever one enum constant is loading into memory, all non constant members (attributes & methods) loading into memory & becoming part of that 'enum constant'.

Whenever object is created, all non static members are loading into memory & becoming part of that object.

Similarly,
Whenever one enum constant is loading into memory, all non static members are loading into memory & becoming part of that enum constant.

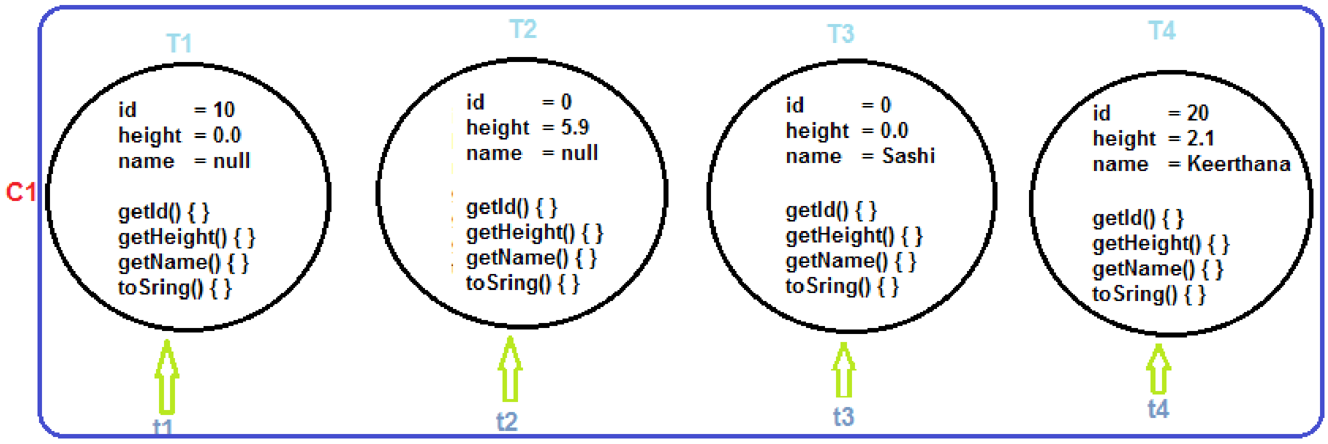
Program 16:

```
	public class Z2 {

		enum C1 {

			T1(10), T2(5.9f), T3("Sashi"), T4(20, 2.1f, "Keerthana");

			private int id;
			private float height;
			private String name;

			C1(int id) {
				System.out.println("C1(int)");
				this.id = id;
			}

			C1(float height) {
				System.out.println("C1(float)");
				this.height = height;
			}

			C1(String name) {
				System.out.println("C1(String)");
				this.name = name;
			}

			C1(int id, float height, String name) {
				System.out.println("C1(int, float, String)");
				this.id = id;
				this.height = height;
				this.name = name;
			}

			public int getId() {
				System.out.print("getId() : ");
				return id;
			}

			public float getHeight() {
				System.out.print("getHeight() : ");
				return height;
			}

			public String getName() {
				System.out.print("getName() : ");
				return name;
			}

			@Override
			public String toString() {
			 return "C1 [id=" + id + ", height=" + height + ", name=" + name + "]";
			}
		}
	public static void main(String[] args) {

		System.out.println("1. =========================");

		C1 t1 = C1.T1;
		C1 t2 = C1.T2;
		C1 t3 = C1.T3;
		C1 t4 = C1.T4;

		System.out.println("2. =========================");

		System.out.println(t1);
		System.out.println(t2);
		System.out.println(t3);
		System.out.println(t4.toString());

		System.out.println("-------------------");

		System.out.println(t1.id);
		System.out.println(t1.name);
		System.out.println("-------------------");

		System.out.println(t2.height);
		System.out.println("-------------------");

		System.out.println(t3.name);
		System.out.println("-------------------");

		System.out.println(t1.getId());
		System.out.println(t1.getHeight());
		System.out.println("-------------------");

		System.out.println(t2.getHeight());
		System.out.println("-------------------");

		System.out.println(t3.getName());
		System.out.println("-------------------");

		System.out.println(t4.getId());
		System.out.println(t4.getName());
	}
}
```

Output:

```
1. =========================
C1(int)
C1(float)
C1(String)
C1(int, float, String)
2. =========================

C1 [id=10, height=0.0, name=null]
C1 [id=0, height=5.9, name=null]
C1 [id=0, height=0.0, name=Sashi]
C1 [id=20, height=2.1, name=Keerthana]

-------------------


10
null
-------------------
5.9
-------------------
Sashi
-------------------
getId() : 10
getHeight() : 0.0
-------------------
getHeight() : 5.9
-------------------
getName() : Sashi
-------------------
getId() : 20
getName() : Keerthana
```

Diagram:



Inbuilt methods of Enum:

1. values() method:

Program 17:

```
public class ValuesMethod {

        enum Days {

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

        public static void main(String[] args) {

                Days allDays[] = Days.values();

                System.out.println("Iterating using Normal for loop");

                for (int i=0; i<allDays.length; i++) {
                        System.out.println(allDays[i]);
                }

                System.out.println("Iterating using Enhanced for loop");

                for (Days day : allDays) {
                        System.out.println(day);
                }
        }
}
```

Output:

```
Iterating using Normal for loop

SUN
MON
TUE
WED
THU
FRI
SAT

Iterating using Enhanced for loop

SUN
MON
TUE
WED
THU
FRI
SAT
```

Explanation:

values() method returns all the enum constants into an array.

To get all the constants use values() method.

2. Ordinal() Method:

Program 18:

```
public class OrdinalMethod {
	enum Directions {

		EAST, WEST, NORTH, SOUTH;
	}

	public static void main(String[] args) {

		Directions d1 = Directions.EAST;
		Directions d2 = Directions.NORTH;

		System.out.println("d1 : " + d1);
		System.out.println("d2 : " + d2);
		System.out.println("------------");

		System.out.println("d1.ordinal() : " + d1.ordinal());
		System.out.println("d2.ordinal() : " + d2.ordinal());
		System.out.println("------------");

		System.out.println("Directions.SOUTH.ordinal() : " +
					Directions.SOUTH.ordinal());
		System.out.println("Directions.WEST.ordinal() : " +
					Directions.WEST.ordinal());
	}
}
```

Output:

```
d1 : EAST
d2 : NORTH
------------

d1.ordinal() : 0
d2.ordinal() : 2
------------

Directions.SOUTH.ordinal() : 3
Directions.WEST.ordinal()  : 1
```

Explanation:

enums are storing from 0th index onwards.

ordinal() method returns the order of the particular enum constant.
It is used for finding out order of the particular enum constant.

ordinal() method return type is 'int'.
enum constants are starting with index 0.

3. valueOf(String) Method:

Program 19:

```
public class ValueOfMethod {
	enum Gender {

		MALE, FEMALE;
	}

	public static void main(String[] args) {

		Gender g1 = Gender.MALE;
		Gender g2 = Gender.FEMALE;

		System.out.println(g1);
		System.out.println(g2);
		System.out.println("-------------");

		System.out.println(Gender.valueOf("MALE"));
			// Accessing in a Static way. This is absolutely correct

		System.out.println(g1.valueOf("FEMALE"));
			// Accessing in a Non Static way. This is absolutely correct

		System.out.println(Gender.valueOf("Hello"));
			// Hello is not available. So it throws "IllegalArgumentException".
	}
}
```

Output:

```
	MALE
	FEMALE
	-------------

	MALE
	FEMALE
	Exception in thread "main" java.lang.IllegalArgumentException:
				No enum const class methodsInEnum.ValueOfMethod$Gender.Hello
```

Program 20:

```
	class ValueOfMethod1{

		public enum Season {

			WINTER, SPRING, SUMMER, FALL
		}

		public static void main(String[] args) {

			for (Season s : Season.values())
				System.out.println("s : " + s);
		}
	}
```

Output:

```
	s : WINTER
	s : SPRING
	s : SUMMER
	s : FALL
```

Inbuilt Enums:

Program 21:

```
public class Mgr1 {

        public static void main(String[] args) {

                // Getting all the Thread states into an array.
                Thread.State states[] = Thread.State.values();
                System.out.println("Thread States are:");
                for (Thread.State state : states) {
                        System.out.println(state);
                }
        }
}
```

Output:

```
Thread States are:
NEW
RUNNABLE
BLOCKED
WAITING
TIMED_WAITING
TERMINATED
```

Method overriding Concept:

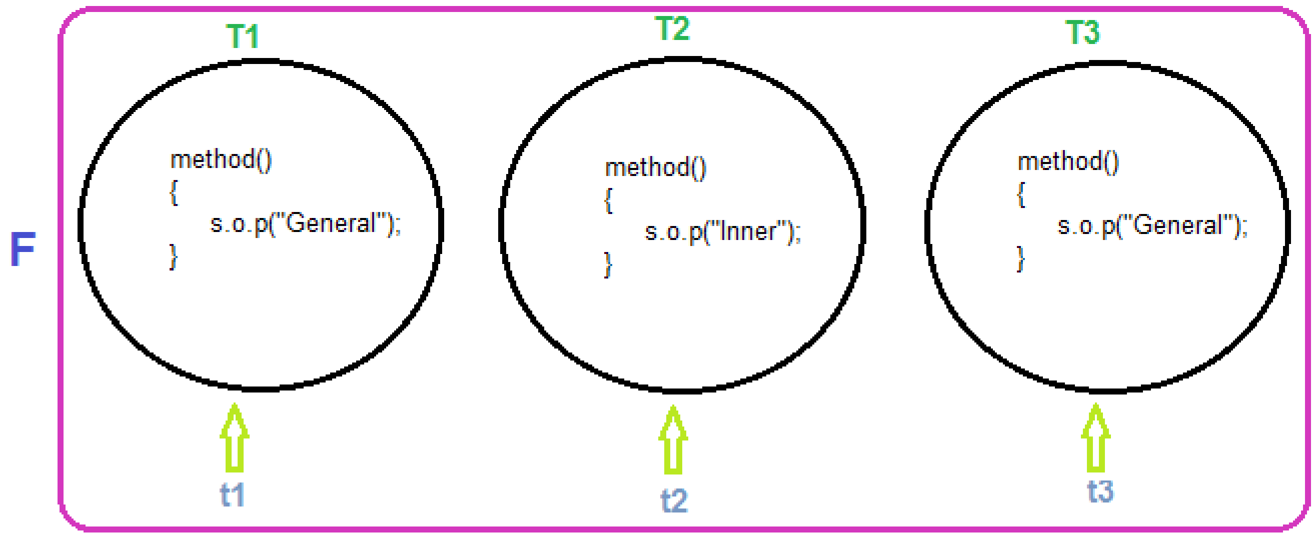
Program 22:

```
public class Z4 {
        enum F {

                T1,
                T2 {
                        void method() {
                                System.out.println("Inner");
                        }
                }, T3;
                void method() {
                        System.out.println("General");
                        return;
                }
        }
        public static void main(String[] args) {
                F t1 = F.T1;
                F t2 = F.T2;
                F t3 = F.T3;

                System.out.println(t1);
                System.out.println(t2);
                System.out.println(t3);
                System.out.println("--------------------");

                t1.method();
                t2.method();
                t3.method();
        }
}
```

Output:

```
T1
T2
T3
--------------------

General
Inner
General
```

Diagram:



Explanation:

enum 'F' contains 3 constants.

T1, & T3 contains one method which is shown below,

```
void method() {
      System.out.println("General");
      return;
}
```

T2 constant contains , specific class body which is shown below,

```
void method() {
      System.out.println("Inner");
}
```

In this class body, U can keep specific things for this particular enum constant.

Program 23:

```
enum Direction {
		// Enum types

		EAST(0) {
			@Override
			public void shout() {
				System.out.println("Direction is East !!!");
			}
		},

		WEST(180) {
			@Override
			public void shout() {
				System.out.println("Direction is West !!!");
			}
		},

		NORTH(90) {
			@Override
			public void shout() {
				System.out.println("Direction is North !!!");
			}
		},

		SOUTH(270) {
			@Override
			public void shout() {
				System.out.println("Direction is South !!!");
			}
		};

		// Constructor
		private Direction(final int angle) {
			System.out.println("Constructor : Direction(int) :  angle = " + angle);
			this.angle = angle;
		}

		// Internal state
		private int angle;

		public int getAngle() {
			return angle;
		}

		// Abstract method which need to be implemented
		public abstract void shout();
	}



	class Mgr {
		public static void main(String[] args) {

			System.out.println("Main Begin");

			Direction d1 = Direction.EAST;

			System.out.println("d1 : " + d1);
			System.out.println("d1.getAngle() : " + d1.getAngle());
			d1.shout();
```

```
                System.out.println("------------------------------------");

                Direction d2 = Direction.SOUTH;

                System.out.println("d2 : " + d2);
                System.out.println("d2.getAngle() : " + d2.getAngle());
                d2.shout();
            }
        }
```

Output:

```
        Main Begin
        Constructor : Direction(int) :  angle = 0
        Constructor : Direction(int) :  angle = 180
        Constructor : Direction(int) :  angle = 90
        Constructor : Direction(int) :  angle = 270

        d1 : EAST
        d1.getAngle() : 0
        Direction is East !!!
        ---------------------------------------------

        d2 : SOUTH
        d2.getAngle() : 270
        Direction is South !!!
```

Why we require arguments to the enum constant?

Consider, if every enum constant having some value, that value is varying from one constant to another constant.

That varying data we can supply as an argument.

Program 23:

```
        public class Mgr2 {
                enum Month {
```

```
                JAN(31), FEB(28), MAR(31), APR(30), MAY(31),
                JUN(31), AUG(31), SEPT(30), OCT(31), NOV(30), DEC(31);

                int days;

                Month(int days) {
                        System.out.println("Month(int) : " + days);
                        this.days = days;
                }

                public int getDays() {
                        return days;
                }
        }

        public static void main(String[] args) {

                Month m1 = Month.FEB;

                System.out.println("Feb. total no. of Days : " + m1.getDays());
                System.out.println("Oct. total no. of Days : " + Month.OCT);
        }
}
```

Output:

```
Main Begin

Month(int) : 31
Month(int) : 28
Month(int) : 31
Month(int) : 30
Month(int) : 31
Month(int) : 31
Month(int) : 31
Month(int) : 30
Month(int) : 31
Month(int) : 30
Month(int) : 31

Feb. total no. of Days : 28
Oct. total no. of Days : OCT
```

Explanation:

Here,

days are common to each enum constant (i.e) each month. That days are varying from one month to another month.

Those varying days, we're supplying as an argument.

Collecting key notes about enum:

- enums are implicitly final subclasses of java.lang.Enum
- if an enum is a member of a class, it’s implicitly static
- new can never be used with an enum, even within the enum type itself
- name and valueOf simply use the text of the enum constants, while toString may be

overridden to provide any content, if desired

- for enum constants, equals and == amount to the same thing, and can be used interchangeably
- enum constants are implicitly public static final
- the order of appearance of enum constants is called their “natural order”, and defines the order used by other items as well : compareTo, iteration order of values , EnumSet, EnumSet.range.
- Constructors for an enum type should be declared as private. The compiler allows non private declares for constructors.
- Since these enumeration instances are all effectively singletons, they can be compared for equality using identity (“==”).
- You can use Enum in Java inside Switch statement like int or char primitive data type

1. Can we create the instance of enum by new keyword?

No, because it contains private constructors only.

2. Can we have abstract method in enum?

Yes, ofcourse! we can have abstract methods and can provide the implementation of these methods.

3. **Can we extend enum types?**

NO, you can not. *Enum types are final by default* and hence can not be extended. Yet, you are free to implement any number of interfaces as you like.