**Enums** OR **Enum Types** OR **Enumeration types** are special data types which are used to represent similar kind of constants. For example, Months (JAN, FEB, MAR, APR . . . . . ), Days (MONDAY, TUESDAY, WEDNESDAY . . . .), Directions (NORTH, SOUTH, WEST, EAST). They have been in other languages like C++ from the beginning. But, in java they are introduced from JDK 1.5. In this post, we will discuss some about 18 points about enums in java which every java developer should be familiar with. It is not a complete Java enum tutorial. It is just a quick overview about the enums in java. You can go through the complete Java Enum tutorial [here](https://javaconceptoftheday.com/java-enums-tutorial-with-examples/).

1) Even though it is not mandatory to declare enum constants with **UPPERCASE** letters, but it is the best practice to do so.

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6 |  |  | enum Grade  {      FIRST,    //Good One        second;   //Bad One  } |

2) Enum types like classes can have **fields**, **constructors** and **methods** along with enum constants.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | enum EnumClass  {      A, B, C;       //Enum constants        int i;      //Enums can have fields        private EnumClass()      {          // Enums can have constructors      }        void methodOfEnum()      {          //Enums can have methods      }  } |

3) Enum constructors are **private** by default. Only private constructors are allowed in enum types. That’s why you can’t instantiate enum types using new operator.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | enum Directions  {      SOUTH, NORTH, EAST, WEST;        private Directions()      {          //Only private constructors are allowed      }        public Directions(int i)      {          //Compile time error, public constructors are not allowed      }  } |

4) Enum constants are created only once for the whole execution. All enum constants are created when you refer any enum constant first time in your code. While creating each enum constant, corresponding constructor is called.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | enum Enums  {      A, B(10), C("ccc", 20);        //No-arg private constructor        private Enums()      {          System.out.println(1);      }        //Private constructor taking one argument        private Enums(int i)      {          System.out.println(2);      }        //Private constructor taking two arguments        private Enums(String s, int j)      {          System.out.println(3);      }  }    public class MainClass  {      public static void main(String[] args)      {          Enums en = Enums.B;   //All enum constants are created while executing this statement.            //While creating each enum constant, corresponding constructor is called            Enums en2 = Enums.C;   //No enum constant is created here.            Enums en3 = Enums.A;   //No enum constant is created here.      }  } |

5) Enum constants must be declared first ahead of fields, constructors and methods (if any).

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | enum Enums  {      int i;        A, B, C;    //Compile time error, enum constants must be declared first.  } |

6) All enum types extend **java.lang.Enum** class by default. As multiple inheritance is not supported in java, they can’t extend any other classes.

7) Enum types can implement any number of interfaces.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | interface InterfaceOne  {      abstract void methodOfInterfaceOne();  }    interface InterfaceTwo  {      abstract void methodOfInterfaceTwo();  }    enum Enums implements InterfaceOne, InterfaceTwo  {      A, B, C;        public void methodOfInterfaceOne()      {          System.out.println("Method Of InterfaceOne");      }        public void methodOfInterfaceTwo()      {          System.out.println("Method Of InterfaceTwo");      }  } |

8) Enum constants can have their own body called **Constant Specific Body**. In that body, you can define fields and methods. But, these methods and fields are visible within the Constant Specific Body in which they are defined.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | enum Enums  {      ONE      {          String s = "ONE";            void methodOfONE()          {              System.out.println(s);          }      },        TWO      {          String s = "TWO";            void methodOfTWO()          {              System.out.println(s);          }      };        void commonMethod()      {          System.out.println("Common method");      }  }    public class MainClass  {          public static void main(String[] args)          {              Enums.ONE.commonMethod();                Enums.TWO.commonMethod();                Enums.ONE.methodOfONE();     //Complie time error, methodOfONE() not visible outisde the ONE's body                Enums.TWO.methodOfTWO();     //Complie time error, methodOfTWO() not visible outisde the TWO's body          }  } |

9) Enum types are **final** by default. They can not be extended by any other types.

10) For every enum type written in a file, **.class** file will be generated after compilation.

11) Enum types can have any number of **static initialization blocks** as well as **instance initialization blocks**.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | enum Enums  {      A, B, C;        static      {          System.out.println("Static Initialization Block");      }        {          System.out.println("Instance Initialization Block");      }  } |

12) As java.lang.Enum class implements Comparable and Serializable interface, all enum types are **Comparable** and **Serializable** by default.

13) We can compare the enum constants using **“==”** operator.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | enum Enums  {      ONE, TWO, THREE, FOUR  }    public class MainClass  {      public static void main(String[] args)      {          System.out.println(Enums.FOUR == Enums.FOUR);     //Output : true            Enums myEnum = Enums.ONE;            System.out.println(myEnum == Enums.ONE);        //Output : true      }  } |

14) You can retrieve the enum constants of any enum type using **values()** method. values() method returns an array of enum constants.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | enum Directions  {      SOUTH, NORTH, EAST, WEST  }    public class MainClass  {      public static void main(String[] args)      {          Directions[] directions = Directions.values();            for (int i = 0; i < directions.length; i++)          {              System.out.println(directions[i]);          }      }  } |

15) Enums provide **type-safety** during compilation. That means you will get compile time error if you try to assign any other values other than the specified enum constants.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | enum Directions  {      SOUTH, NORTH, EAST, WEST  }    public class MainClass  {      public static void main(String[] args)      {          Directions direction1 = RIGHT;    //Compile time error            //You can't assign any other values other then SOUTH, NORTH, EAST, WEST      }  } |

16) You can define enum types outside a class or inside a class but not inside a method or block.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | enum EnumOne  {      A, B, C        //Enum outside the class  }    public class MainClass  {      enum EnumTwo      {          A, B, C            //Enum inside the class      }        public static void main(String[] args)      {          enum EnumThree          {              A, B, C                //Compile Time Error, can't define enum inside a method          }      }  } |

17) **ordinal()** method is used get the order of an enum constant in an enum type.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | enum Directions  {      SOUTH, NORTH, EAST, WEST  }    public class MainClass  {      public static void main(String[] args)      {          System.out.println(Directions.EAST.ordinal());    //Output : 2            System.out.println(Directions.NORTH.ordinal());   //Output : 1      }  } |

18) Enums are mostly used when you want to allow limited set of options which remain constant for whole execution and you know all possible options at compile time itself. For example, **choices on a menu** or **options of a combobox**.