What is Enum is java:

In Java, an enum (short for enumeration) is a special type that represents a fixed set of constant values. It allows you to define your own data type with a limited number of predefined values, making your code more readable, maintainable, and type-safe. Enumerations are declared using the enum keyword.

Here are some important aspects of Java enums:

1. **Declaration and Syntax**: To declare an enum, you use the enum keyword followed by the enum name and a list of constant values within curly braces.

Syntax:

```
enum EnumName {
   VALUE1,
   VALUE2,
   // ...
}
```

Example:

```
enum DayOfWeek {
   MONDAY,
   TUESDAY,
   WEDNESDAY,
   THURSDAY,
   FRIDAY,
   SATURDAY,
   SUNDAY
}
```

2. **Enum Constants:** The constant values defined within an enum are called enum constants. They are implicitly declared as public, static, and final. Each constant represents an instance of the enum type.

Example usage:

```
DayOfWeek day = DayOfWeek.MONDAY;
```

3. **Methods and Properties:** Enums can have methods and properties associated with each constant value. You can define custom behaviors or retrieve additional information related to each enum constant.

Example:

```
enum DayOfWeek {
  MONDAY("Mon"),
  TUESDAY("Tue"),
  WEDNESDAY("Wed"),
  // ...
  private String abbreviation;

  DayOfWeek(String abbreviation) {
    this.abbreviation = abbreviation;
  }
  public String getAbbreviation() {
    return abbreviation;
  }
}
```

Usage:

```
DayOfWeek day = DayOfWeek.MONDAY;
System.out.println(day.getAbbreviation()); // Output: Mon
```

4. **Switch Statements**: Enums are commonly used with switch statements to perform different actions based on the enum constant. The switch statement can directly operate on the enum type, simplifying the code and providing compile-time safety.

Example:

```
DayOfWeek day = DayOfWeek.MONDAY;
switch (day) {
  case MONDAY:
  case TUESDAY:
  case WEDNESDAY:
  case THURSDAY:
  case FRIDAY:
      System.out.println("It's a weekday");
      break;
  case SATURDAY:
  case SUNDAY:
      System.out.println("It's a weekend");
      break;
}
```

5. **Iterating over Enum Values**: You can iterate over all the enum constants using the values() method, which returns an array containing all the enum values.

Example:

```
for (DayOfWeek day : DayOfWeek.values()) {
    System.out.println(day);
}
```

6. **Enum Inheritance:** Enums can implement interfaces and inherit from other classes. However, enums cannot be subclassed.

Example:

```
interface Printable {
    void print();
}
enum Color implements Printable {
    RED,
    GREEN,
    BLUE;
    @Override
    public void print() {
        System.out.println("Printing color: " + this.name());
    }
}
```

Enums provide a convenient and type-safe way to define and work with a fixed set of values. They eliminate the need for using magic numbers or strings in your code, making it more robust and readable. Enums are widely used for representing a range of values, such as days of the week, states, error codes, and more.

Equality of Enum:

In Java, you can check the equality of enum constants using the == operator or the equals() method. Enums are reference types, so these approaches compare the references to determine if the enum constants are equal.

Here's how you can check the equality of enum constants:

1. **Using the == operator:** The == operator compares the references of the enum constants. It returns true if the references are pointing to the same enum constant, and false otherwise.

Example:

```
enum Color {
   RED,
   GREEN,
   BLUE
}
Color color1 = Color.RED;
Color color2 = Color.RED;
Color color3 = Color.GREEN;
System.out.println(color1 == color2);  // Output: true
System.out.println(color1 == color3);  // Output: false
```

2. **Using the equals() method:** The equals() method compares the references of the enum constants. It is inherited from the Object class and can be overridden by the enum class to provide custom equality logic. By default, the equals() method behaves the same as using the == operator for enum constants.

Example:

```
enum Color {
   RED,
    GREEN,

    BLUE

}

Color color1 = Color.RED;

Color color2 = Color.RED;

Color color3 = Color.GREEN;


System.out.println(color1.equals(color2));  // Output: true

System.out.println(color1.equals(color3));  // Output: false
```

It's important to note that comparing enum constants using the == operator or the equals() method is typically sufficient. However, if you have overridden the equals() method in your enum class to provide custom equality logic, make sure to consider that during the comparison.

In most cases, checking enum equality using the == operator is preferred, as it is more concise and efficient. However, if you want to use the equals() method for consistency with other object types, you can do so as well.

Example:

```
package util.blocks;

public enum WeekDays {
   SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

   //public static WeekDays SUNDAY = new WeekDays("SUNDAY")

   private int dayOfTheWeek;

   WeekDays(){
      System.out.println("Calling constructor for value "+name());
//       switch (name()){
//          case "SUNDAY" :
//             this.dayOfTheWeek = 1;
//             break;
//          case "SATURDAY" :
//             this.dayOfTheWeek = 7;
//             break;
//       }
      if(name().equals("SUNDAY")){
         this.dayOfTheWeek = 1;
      }
```

```
    }

    public int getDayOfTheWeek() {
        return dayOfTheWeek;
    }

    public static WeekDays[] getHolidaysOfWeek(){
        WeekDays[] arr = new WeekDays[2];
        arr[0] = SUNDAY;
        arr[1] = SATURDAY;
        return arr;
    }

    public String getShortcut(){
        switch (name()){
            case "SATURDAY" :
                return "Sat";
            case "SUNDAY" :
                return "Sun";
        }
        return null;
    }
}
```

```
package util.blocks;

public class EnumDemo {
    private static final String MONDAY = "MONDAY";

    public static void main(String[] args) {
        System.out.println(WeekDays.MONDAY);
        System.out.println(WeekDays.TUESDAY);

        //1st operation
        WeekDays[] weekDays = WeekDays.values();

        //2nd operation
        WeekDays monday = WeekDays.valueOf("MONDAY");
        System.out.println(monday);

        //3rd operation
        System.out.println(monday == WeekDays.MONDAY);
        System.out.println(monday.equals(WeekDays.MONDAY));
        //4th operation
        switch (monday){
            case MONDAY :
                System.out.println("Monday");
                break;

            case TUESDAY :
                System.out.println("TUESDAY");
                break;
        }


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

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

        //5th operation
        WeekDays[] holidays = WeekDays.getHolidaysOfWeek();

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

        //6th operation
        System.out.println("Weekday shortcut for saturday is : "+WeekDays.SATURDAY.getShortcut());
        System.out.println("Weekday shortcut for sunday is : "+WeekDays.SUNDAY.getShortcut());

        //7th operation
        System.out.println("Day of the week for Saturday: "+WeekDays.SATURDAY.getDayOfTheWeek());
        System.out.println("Day of the week for Sunday: "+WeekDays.SUNDAY.getDayOfTheWeek());
    }
}
```