



ATME

College of Engineering

GOLD

QS I-GAUGE
INDIAN COLLEGE RANKING



Object-Oriented Programming with Java



Module 5: Enumerations, Type Wrappers, Autoboxing and Generics

Enumerations

An **enumeration (enum)** is a special data type used to define a collection of **named constants**

Syntax

```
enum enumName {  
    CONSTANT1, CONSTANT2, CONSTANT3  
}
```

```
enum Direction {  
    NORTH, SOUTH, EAST, WEST  
}
```

OUTPUT
NORTH

```
class Demo {  
    public static void main(String[] args) {  
        Direction d = Direction.NORTH;  
  
        System.out.println(d);  
    }  
}
```

Example using switch

```
enum Day {  
    MONDAY, TUESDAY, WEDNESDAY  
}  
  
class Test {  
    public static void main(String[] args) {  
        Day d = Day.MONDAY;  
  
        switch(d) {  
            case MONDAY:  
                System.out.println("Start of the week");  
                break;  
            case TUESDAY:  
                System.out.println("Tuesday");  
                break;  
  
            case WEDNESDAY:  
                System.out.println("Wednesday");  
                break;  
        }  
    }  
}
```

values() Method

The values() method returns **all constants of an enum** in the order in which they are declared.

```
enum Color {  
    RED, GREEN, BLUE  
}
```

```
class Demo {  
    public static void main(String[] args) {  
  
        for(Color c : Color.values()) {  
            System.out.println(c);  
        }  
    }  
}
```

valueOf() Method

The valueOf() method converts a **string into the corresponding enum constant**.

```
enum Color {  
    RED, GREEN, BLUE  
}
```

```
class Demo {  
    public static void main(String[] args) {  
  
        Color c = Color.valueOf("GREEN");  
  
        System.out.println(c);  
    }  
}
```

Type Wrappers

Java provides **wrapper classes** to represent primitive data types as objects

Primitive	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean



Character Wrapper

The wrapper class for char is **Character**.

Character ch = 'A';

```
class Demo {  
    public static void main(String[] args) {  
  
        char ch = 'A';  
  
        System.out.println(Character.isLetter(ch));  
  
        System.out.println(Character.isUpperCase(ch));  
    }  
}
```

Autoboxing

Autoboxing is the automatic conversion of a primitive value into its corresponding wrapper object.

Example

```
int x = 10;  
Integer obj = x;
```

Java automatically converts:

int → Integer

This is called **autoboxing**.

Unboxing

Unboxing is the automatic conversion of a wrapper object into its corresponding primitive value.

```
Integer obj = 50;
```

```
int x = obj;
```

Autoboxing and Methods

Autoboxing is useful when a method expects a wrapper object.

```
class Demo {  
  
    static void display(Integer x) {  
        System.out.println(x);  
    }  
  
    public static void main(String[] args) {  
  
        int num = 25;  
  
        display(num);  
    }  
}
```

Generics

Generics allow us to create classes, methods, and interfaces that work with different data types while providing **type safety**.

Generics were introduced in Java 5.

Without Generics

```
class Test {  
    Object value;  
  
    Test(Object value) {  
        this.value = value;  
    }  
}
```

With Generics

```
class Test<T> {  
    T value;  
  
    Test(T value) {  
        this.value = value;  
    }  
  
    T getValue() {  
        return value;  
    }  
}
```

A Simple Generics Example

```
class Gen<T> {  
  
    T value;  
  
    Gen(T value) {  
        this.value = value;  
    }  
  
    void showType() {  
        System.out.println(value.getClass().getName());  
    }  
  
    T getValue() {  
        return value;  
    }  
}
```

```
class Demo {  
    public static void main(String[] args) {  
  
        Gen<Integer> obj1 = new Gen<>(100);  
        Gen<String> obj2 = new Gen<>("Hello");  
  
        System.out.println(obj1.getValue());  
        System.out.println(obj2.getValue());  
    }  
}
```

Generic Class with Two Type Parameters

A generic class can have **more than one type parameter**.

```
class Pair<T, U> {  
  
    T first;  
    U second;  
  
    Pair(T first, U second) {  
        this.first = first;  
        this.second = second;  
    }  
  
    void display() {  
        System.out.println(first);  
        System.out.println(second);  
    }  
}
```

Using the class:

```
class Demo {  
    public static void main(String[] args) {  
  
        Pair<Integer, String> p =  
            new Pair<>(101, "Divya");  
  
        p.display();  
    }  
}
```

```
class Sample<T> {
```

```
    T data;
```

```
    Sample(T data) {  
        this.data = data;  
    }
```

```
    T getData() {  
        return data;  
    }  
}
```



A T M E
College of Engineering

GOLD

QS I-GAUGE
INTEGRITY & INNOVATION



THANK YOU