เรื่องที่ 8 Wrapper Classes Math Class, Enum

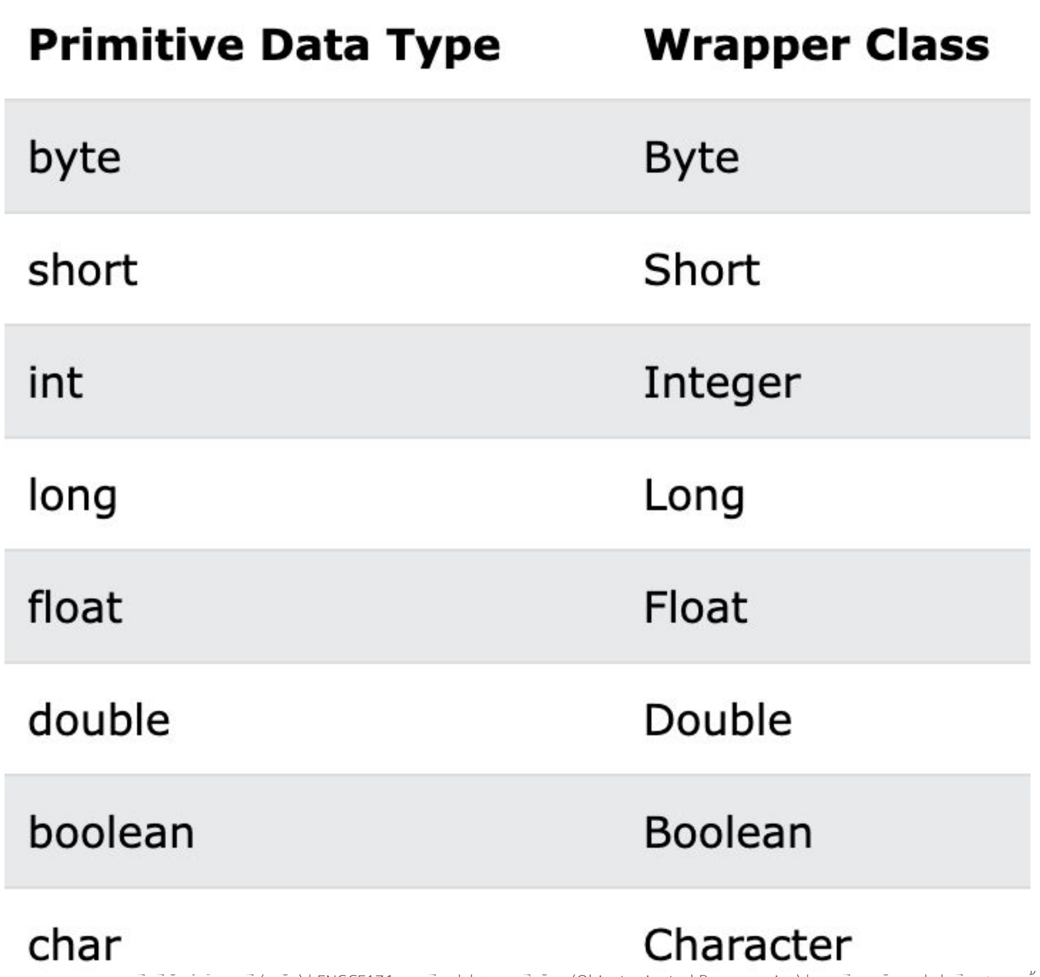
ENGCE174 การเขียนโปรแกรมเชิงวัตถุ (Object-oriented programming)

อ.กิตตินันท์ น้อยมณี (mr.kittinan@rmutl.ac.th)

Java Wrapper Classes

Wrapper classes provide a way to use primitive data types (`int`, `boolean`, etc..) as objects.

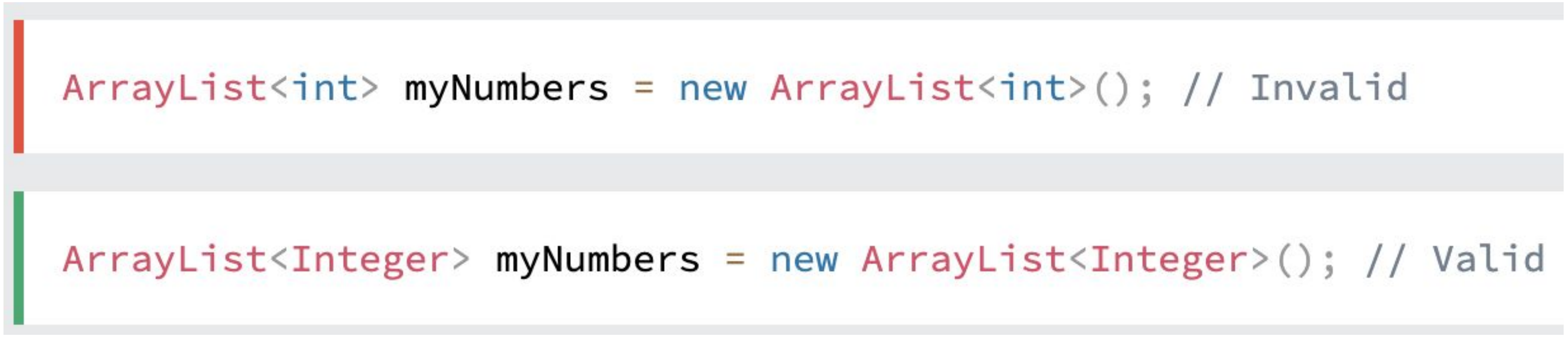
The table below shows the primitive type and the equivalent wrapper class:



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

Java Wrapper Classes

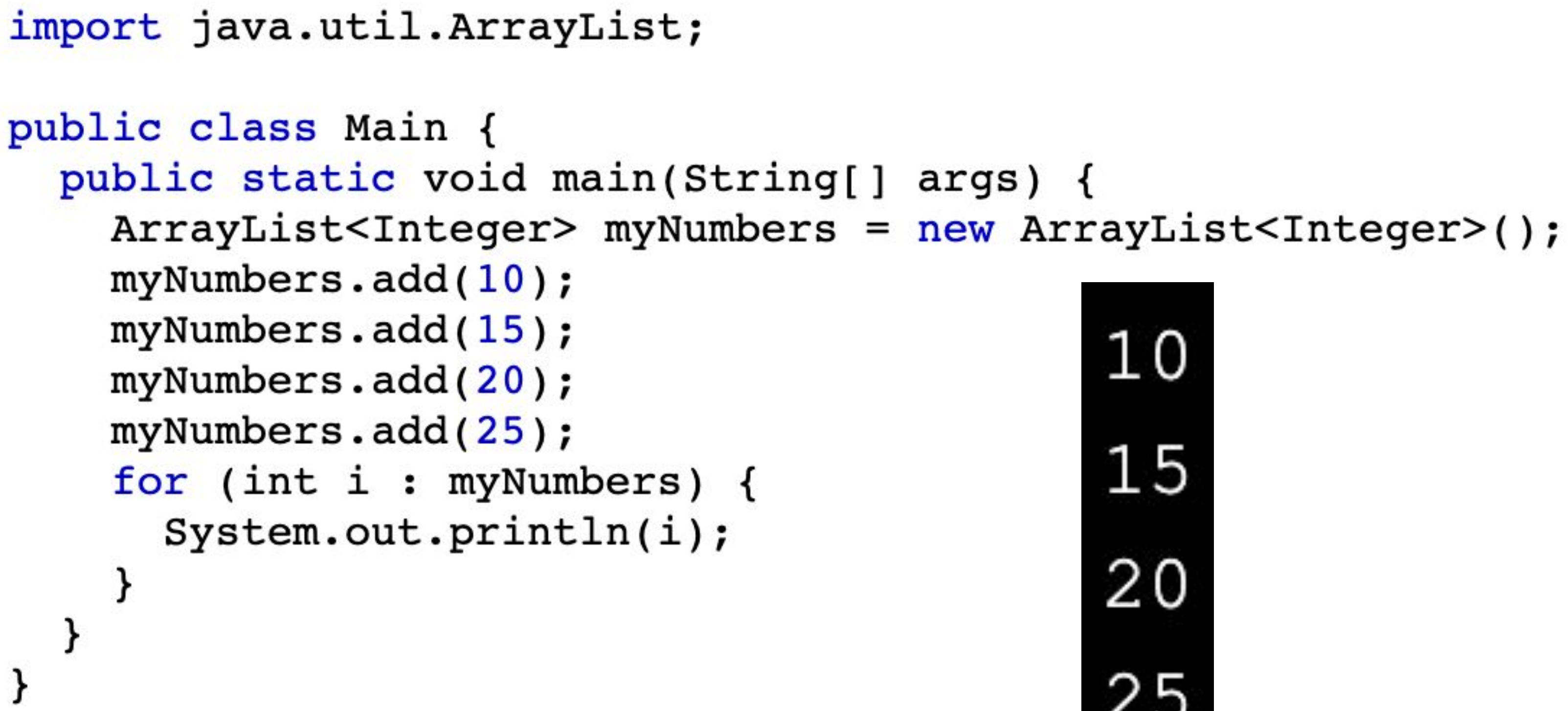
Sometimes you must use wrapper classes, for example when working with Collection objects, such as `ArrayList`, where primitive types cannot be used (the list can only store objects):



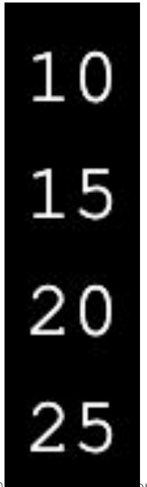
```
ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid
```

```
ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid
```

Java Wrapper Classes



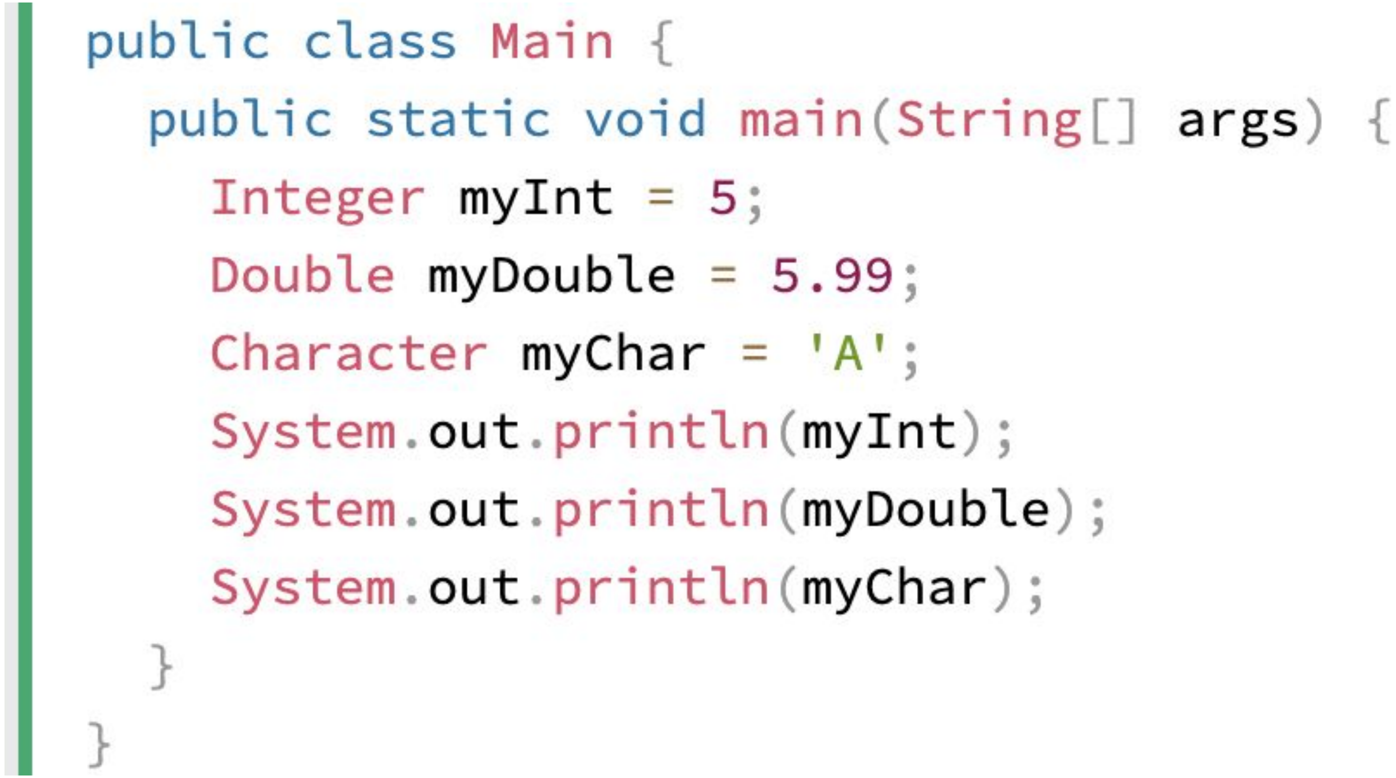
```java
import java.util.ArrayList;

public class Main {
  public static void main(String[] args) {
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();
    myNumbers.add(10);
    myNumbers.add(15);
    myNumbers.add(20);
    myNumbers.add(25);
    for (int i : myNumbers) {
      System.out.println(i);
    }
  }
}
```



```
10
15
20
25
```

Creating Wrapper Objects

To create a wrapper object, use the wrapper class instead of the primitive type. To get the value, you can just print the object:



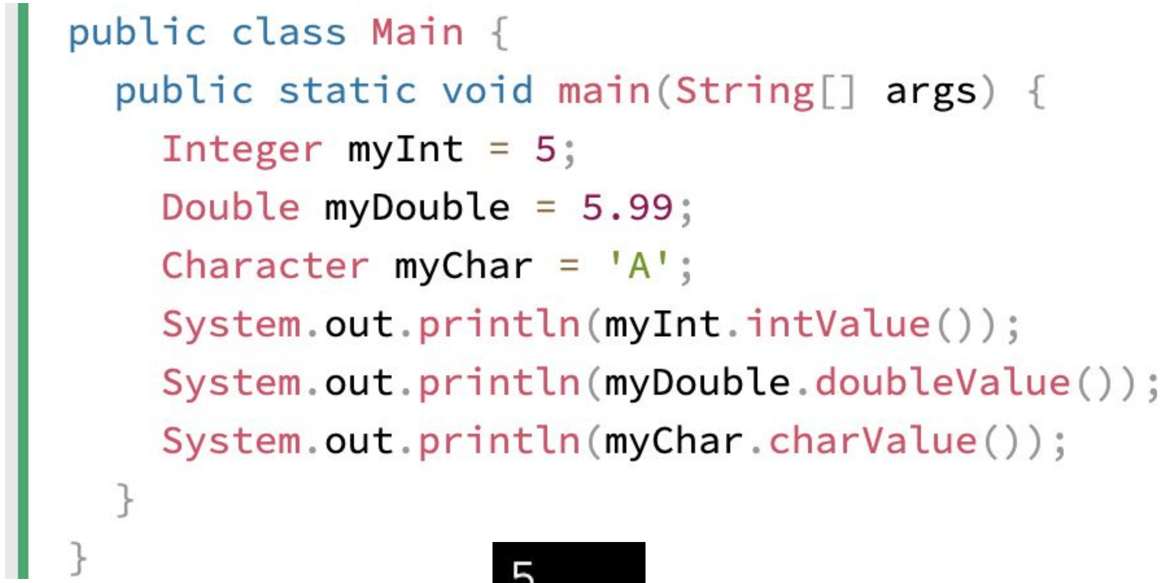
```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }
}
```

Creating Wrapper Objects

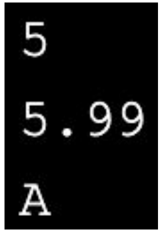
Since you're now working with objects, you can use certain methods to get information about the specific object.

For example, the following methods are used to get the value associated with the corresponding wrapper object: `intValue()`, `byteValue()`, `shortValue()`, `longValue()`, `floatValue()`, `doubleValue()`, `charValue()`, `booleanValue()`.

This example will output the same result as the example above:



```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt.intValue());
    System.out.println(myDouble.doubleValue());
    System.out.println(myChar.charValue());
  }
}
```

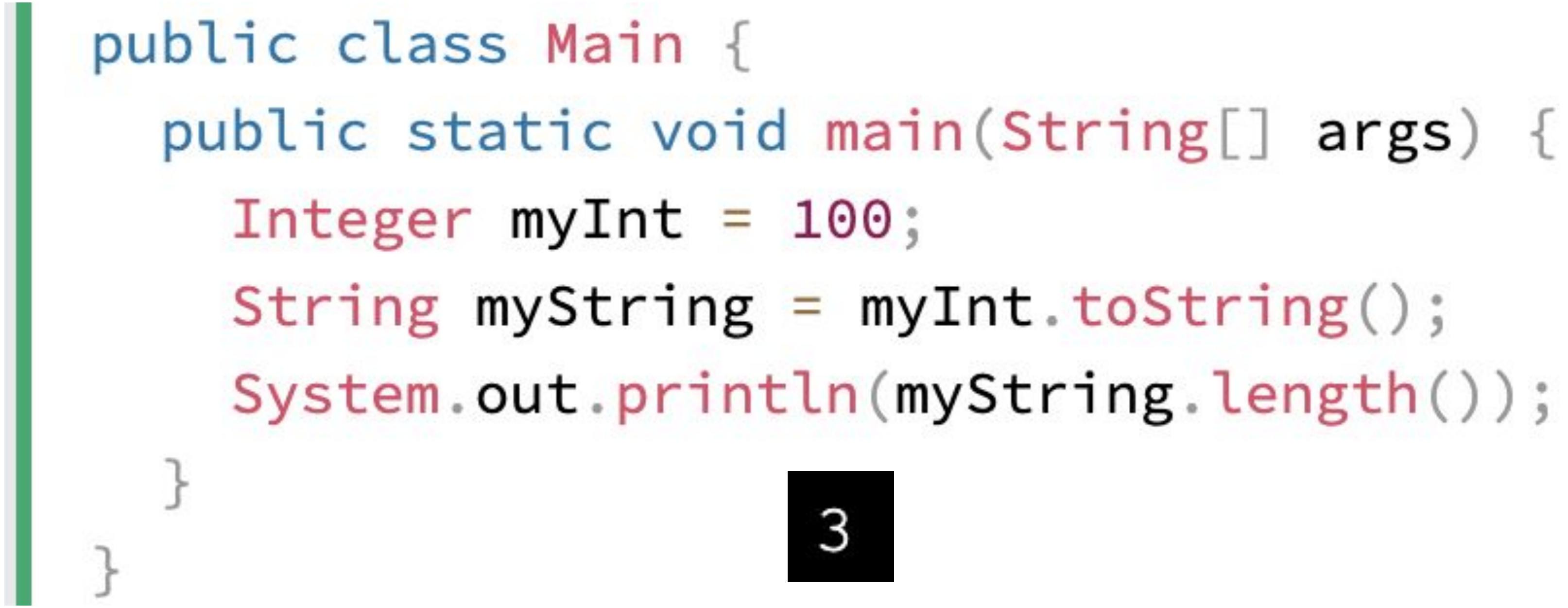


```
5
5.99
A
```

Creating Wrapper Objects

Another useful method is the `toString()` method, which is used to convert wrapper objects to strings.

In the following example, we convert an `Integer` to a `String`, and use the `length()` method of the `String` class to output the length of the "string":



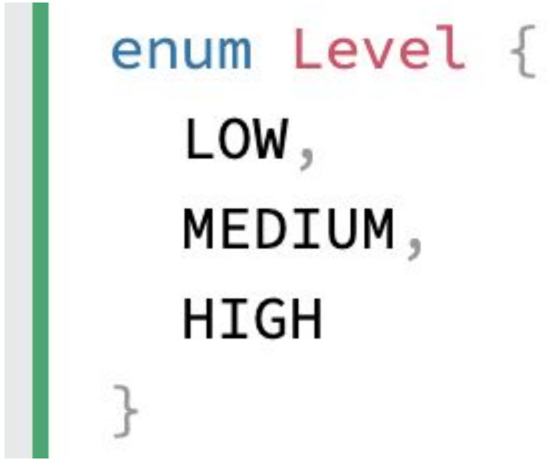
```java
public class Main {
  public static void main(String[] args) {
    Integer myInt = 100;
    String myString = myInt.toString();
    System.out.println(myString.length());
  }
}
```

```
3
```

Enums

An enum is a special "class" that represents a group of constants (unchangeable variables, like final variables).

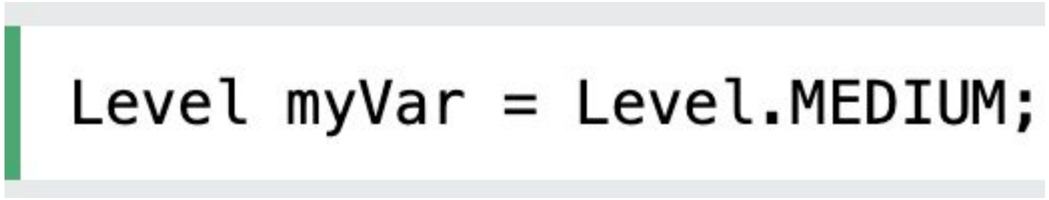
To create an enum, use the enum keyword (instead of class or interface), and separate the constants with a comma. Note that they should be in uppercase letters:



```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}
```

Enums

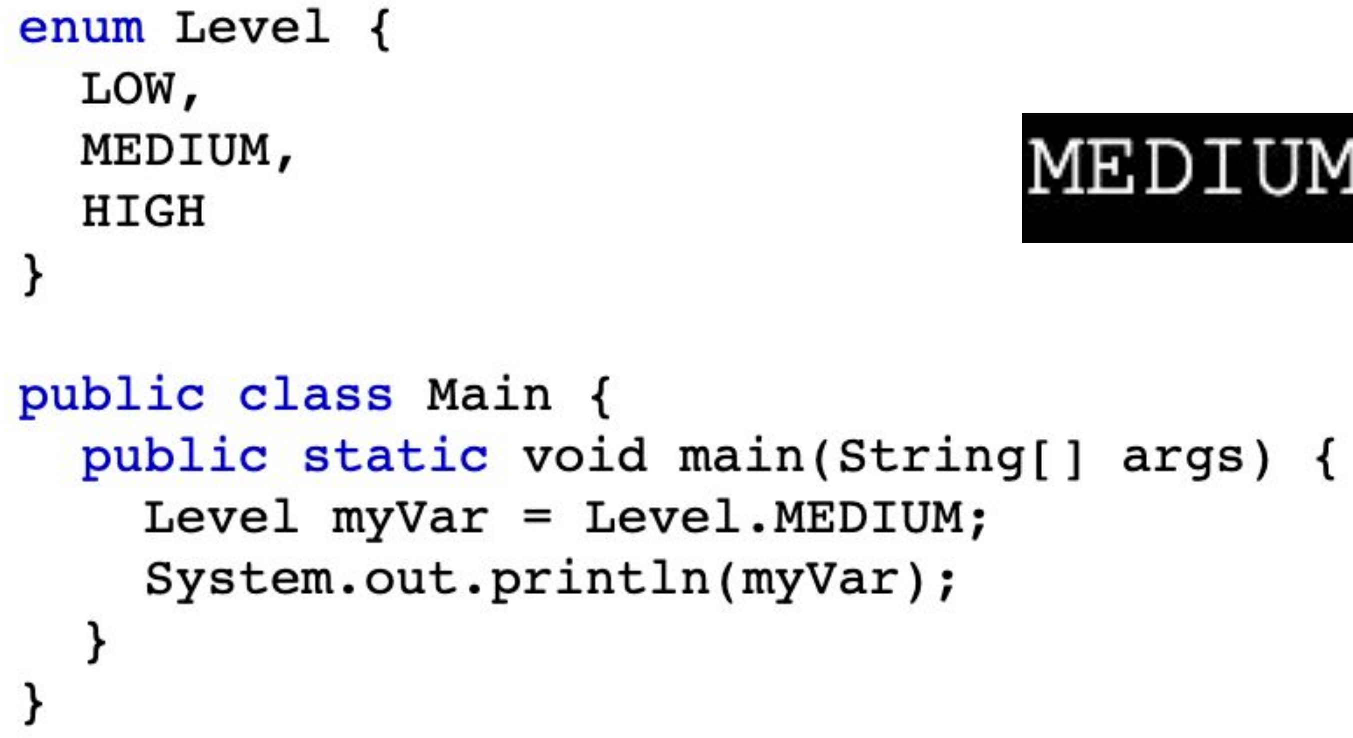
You can access `enum` constants with the dot syntax:



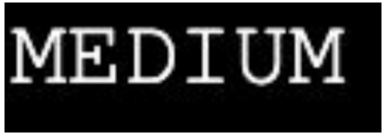
```
Level myVar = Level.MEDIUM;
```

Enums

Enum is short for "enumerations", which means "specifically listed".



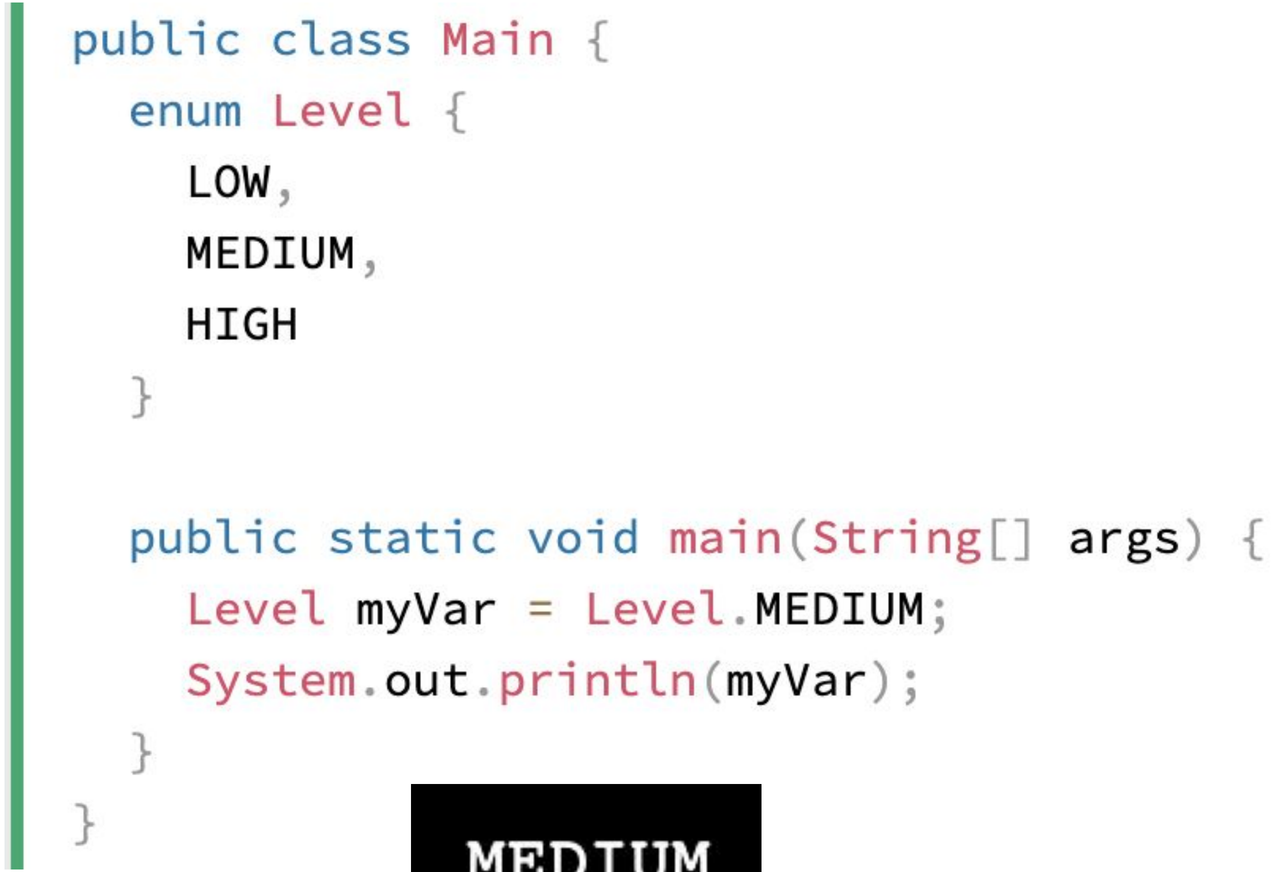
```
enum Level {
  LOW,
  MEDIUM,
  HIGH
}

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
    System.out.println(myVar);
  }
}
```



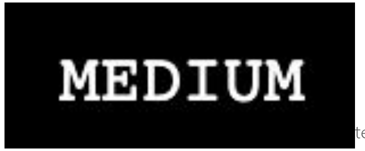
```
MEDIUM
```

Enum inside a Class

You can also have an `enum` inside a class:



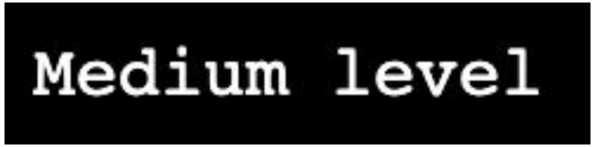
```java
public class Main {
  enum Level {
    LOW,
    MEDIUM,
    HIGH
  }

  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;
    System.out.println(myVar);
  }
}
```



```
MEDIUM
```

Enum in a Switch Statement

Enums are often used in `switch` statements to check for corresponding values:

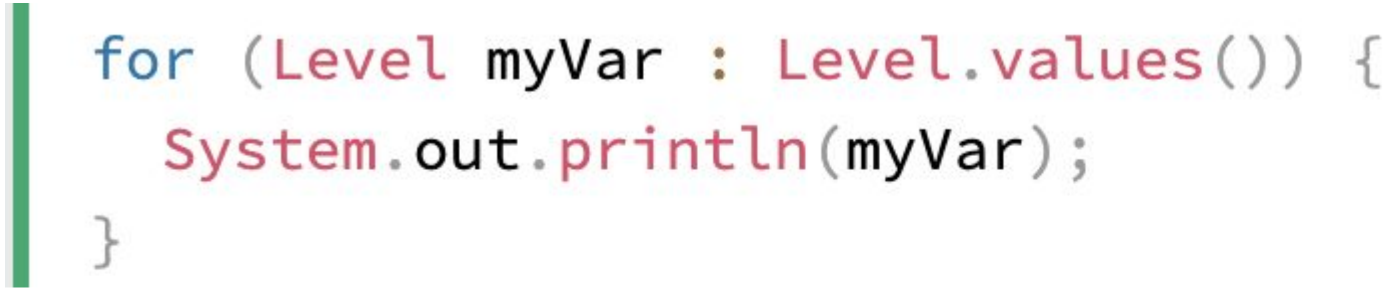


```
Medium level
```

```java
enum Level {
  LOW,
  MEDIUM,
  HIGH
}

public class Main {
  public static void main(String[] args) {
    Level myVar = Level.MEDIUM;

    switch(myVar) {
      case LOW:
        System.out.println("Low level");
        break;
      case MEDIUM:
         System.out.println("Medium level");
        break;
      case HIGH:
        System.out.println("High level");
        break;
    }
  }
}
```

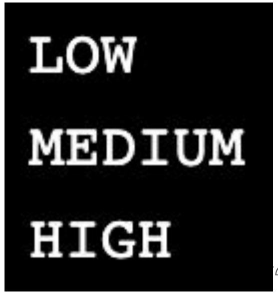
Loop Through an Enum

The enum type has a `values()` method, which returns an array of all enum constants. This method is useful when you want to loop through the constants of an enum:



```
for (Level myVar : Level.values()) {
  System.out.println(myVar);
}
```

The output will be:



```
LOW
MEDIUM
HIGH
```

Loop Through an Enum

Difference between Enums and Classes

An `enum` can, just like a `class`, have attributes and methods. The only difference is that enum constants are `public`, `static` and `final` (unchangeable - cannot be overridden).

An `enum` cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

Why And When To Use Enums?

Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.