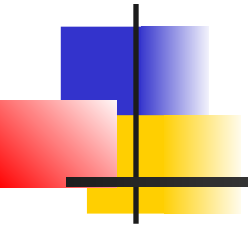


enum: Enumerated Type

An "enum" is a type with a fixed set of elements.

What is "enum"

"enum" (enumeration) defines a new data type that has a fixed set of values.

Example: an enum named Size with 3 fixed values:

```
public enum Size {
        SMALL,
        MEDIUM,
        LARGE;
}
```

Define an enum

An "enum" defines a type, just like a class.

```
public enum Size {
        SMALL,
        MEDIUM,
        LARGE;
}
```

List each element followed by a COMMA, except last one.

```
// correct usage
Size size = Size.SMALL;
// enum can be parameter
public void setSize(Size s)...
// Illegal: no new instances
Size size = new Size( );
```

Using an enum

enum type can be a variable, parameter, or return type

```
// can be type type of an attribute or local var:
private Size mysize;
// can be parameter:
public void setSize(Size size) { this.mysize = size; }
// can compare values using ==
public double getPrice( ) {
    if (mysize == Size.SMALL) return 20.0;
    if (mysize == Size.MEDIUM) return 30.0;
    if (mysize == Size.LARGE) return 40.0;
    else return 0;  // possible if size is null
}
```

Why use enum?

Compiler can check if values are legal.

Better Type Safety

Example: suppose the Coffee size is a *String*.

```
public class Coffee {
  private String size;
  // size must be "small","medium","large"
  public Coffee( String size ) {
    this.size = size;
  }
```

NO ERROR

```
Coffee sbucks = new Coffee( "Grande" );
```

Coffee class with Size enum

```
public class Coffee {
   private Size size;
   // size must be an enum member
   public Coffee( Size size ) {
      this.size = size;
   }
```

ERROR

```
Coffee sbucks = new Coffee( "Grande" );
```

Another Example: Font class

The Font constructor is:

```
new Font(String name, int style, int size)
```

Font.PLAIN = 0
Font.BOLD = 1
Font.ITALIC = 2

Correct: Arial BOLD, size 20pt

```
Font font = new Font("Arial", 1, 20);
```

Incorrect, but **no error** at compile or runtime:

```
Font font = new Font("Arial", 20, 1);
```

Result is a tiny font with pointsize 1

Design a Better Font Class

If the font style were an enum, this error would not occur

```
public enum FontStyle {
        PLAIN,
        BOLD,
        ITALIC,
        BOLD_ITALIC;
}
```

```
Font font =
    new Font("Arial", FontStyle.BOLD, 20);
```

Applying enum to Coffee

```
public class Coffee {
private Size size;
public Coffee( Size size ) {
      this.size = size;
}

public double getPrice( ) {
   switch( size ) {
        case SMALL: return 20.0;
        case MEDIUM: return 30.0;
        case LARGE: return 40.0;
        default: return 0;
   }
}
```

Use of enum

1. You can declare a variable of an **enum** type:

```
Size size;  // size is of type "Size"
```

2. You can assign a value to an enum variable:

```
Size s = Size.SMALL;
```

3. You can compare values using ==

```
if ( size == Size.SMALL ) price = 20.0;
```

4. You can use enum in switch.

```
switch( size ) { case SMALL: ... }
```

5. You can print the values (implicit toString()).

```
System.out.println("Size is " + size );
```

values( ) method

- Every **enum** has a **values( )** method that returns an *array* of the members of the **enum**.

```
> Size.values( )

    Size[ ]{ SMALL, MEDIUM, LARGE }
```

Automatic conversion to String with same name as enum elements:

```
> for( Size s: Size.values() )
   System.out.println( s );
 SMALL
 MEDIUM
 LARGE
```

Other Enum methods

- Every enum also has these methods

compareTo(E other)	> Size.SMALL.compareTo(Size.LARGE) -2
name( )	> Size.SMALL.name() "SMALL"
valueOf(String)	Get enum member with the String value: > Size.valueOf("LARGE") (Size) Size.LARGE
toString()	Returns declared name as String, like name( ) > Size.SMALL.toString() "SMALL"

enum can have attributes (properties)

- **enum** can have properties and methods, just like a class.

Example: add a price attribute to Size enum.

```
enum Size {
   SMALL(20.0),
   MEDIUM(30.0),
   LARGE(40.0);
   private final double price;
   /** constructor sets the price */
   private Size(double price) {
            this.price = price;
       }
   public double getPrice() { return price; }
```

Declare attributes ***after*** the enum members.

Private Constructor

- **enum** can have constructors, but they must be private.
- Private is the default for "enum" constructors.

```
enum Size {
   SMALL(20),
   MEDIUM(30),
   LARGE(40);
   public final double price;
   Size(double price) { this.price = price; }
    public double getPrice() { return price; }
}
```

"private" by default.

Using enum Attributes

We can use enum price attribute to simplify getPrice.

```
class Coffee {
  private Size size;
  public Coffee( Size size ) { ... }

  public double getPrice() {
    return size.getPrice();
  }
```

Attributes should make sense

enum represent *constants*. enum can have multiple uses.

But price is something likely to vary or change.

```
class Pizza {
  Size size;  // size of the pizza
  double getPrice() {
    return size.getPrice();
  }
```

Arrrrgh! This is the coffee price!

enum for Length

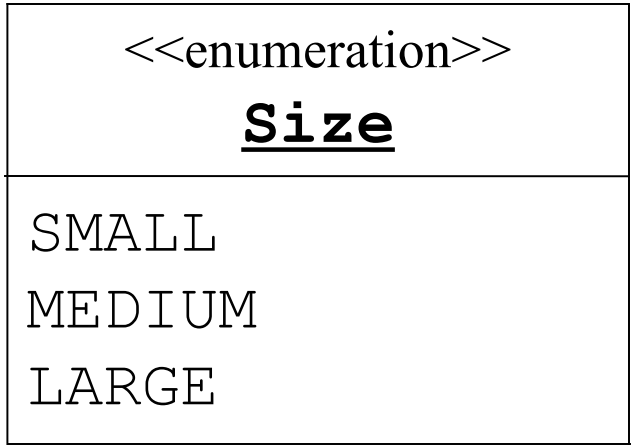
Use enum for values of length in a UnitConverter

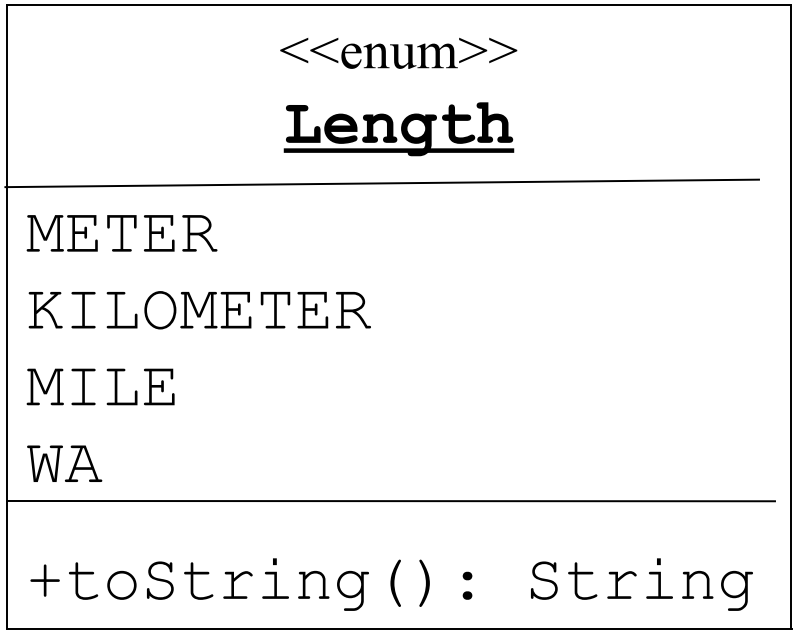
```
public enum Length {
   METER("meter", 1.0),
   KILOMETER("km", 1000.0),
   MILE("mile", 1609.344),
   WA("wa", 2.0);
   public final double value;
   public final String name;
   public Length( String name, double val ) {
      this.value = val; this.name = name; }
   public String toString() { return name; }
}
```

Attributes as public constants

UML for Enumeration

enum with no methods:





UML Distilled has notation for enum in UML.