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: Coffee has a size.

The size can be "small", "medium", or "large" -- but no other values.

```
Coffee java = new Coffee( SMALL );
```

What we want. But how to do in Java?

Define an enum

An "enum" defines a type, like "class" or "interface".

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

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

```
// correct usage
Size size = Size.SMALL;
// illegal (no new instance)
Size size = new Size( );
```

Using an enum

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

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

Why "enum"?

Compiler can check if values are legal or not.

Avoids Programming Errors

Better Type Safety

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

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

NO ERROR!

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

Why "enum"? Font class

The font constructor is:

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

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

Correct:

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

Incorrect, but no error at compile or runtime:

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

Result is a <u>tiny</u> font with pointsize = 1 (= Font.BOLD*)*

Applying enum to Coffee

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

 public double getPrice( ) {
    switch( size ) {
    case Size.SMALL: return 20.0;
    case Size.MEDIUM: return 30.0;
    case Size.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 );
```

enum 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"
ordinal()	Index of enum member in the set: > int k = SMALL.ordinal() 0

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 int 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 int price;
    Size( int price) { this.price = price; }
     public int 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();
   }
```

if size is null then throw IllegalArgumentException

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

Length enum

1. Length values don't change -- good use of property.

2. Attribs are public final as *convenience* for programmer:

```
// convert 2.5 kilometers to miles
double km = 2.5;
double mi = km * Length.KILOMETER.value /
            Length.MILE.value ;
  // don't need to call getValue()
```

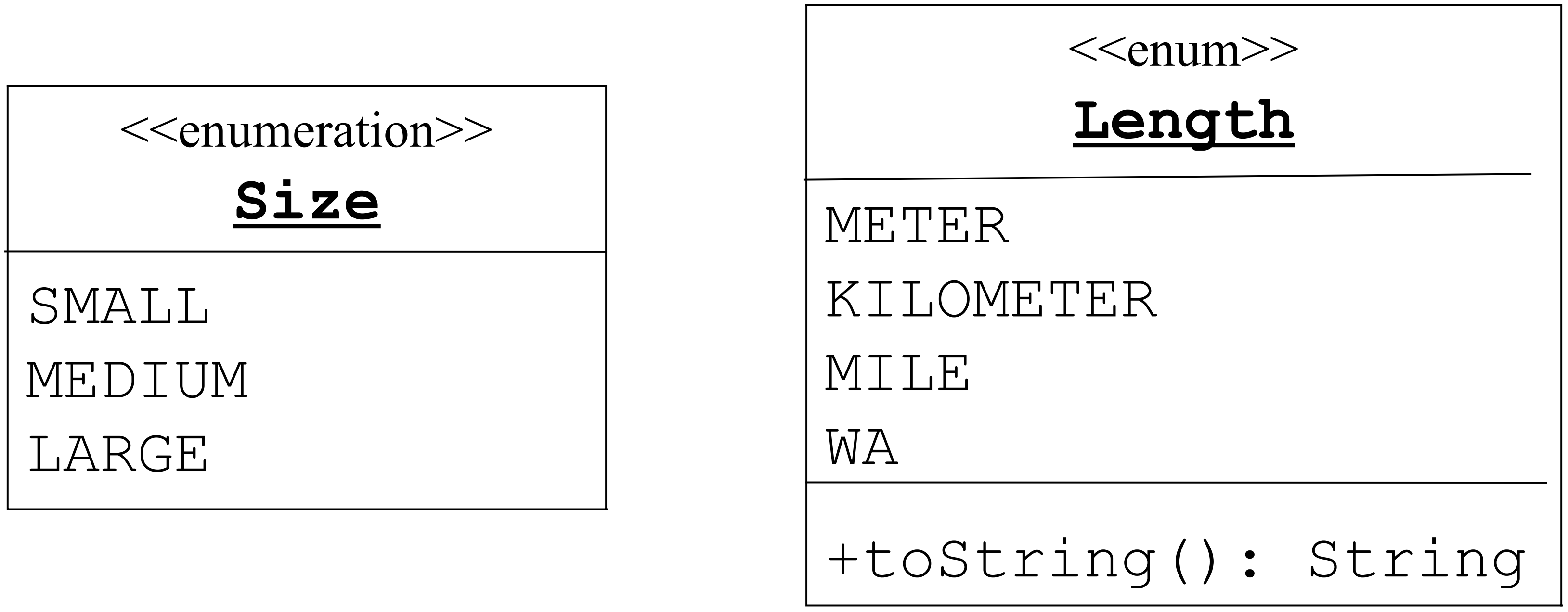
3. Define **`toString()`** in Length for prettier output:

Length without `toString`:   Length.MILE ==> "MILE"

Length with `toString`:   Length.MILE ==> "Mile"

UML for Enumeration

enum with no methods:



UML Distilled has notation for enum in UML.