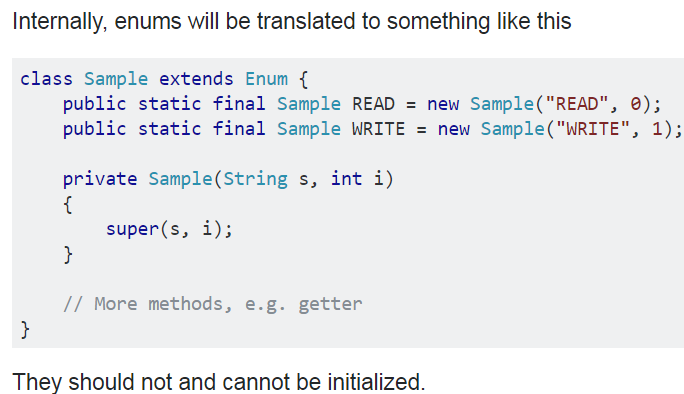
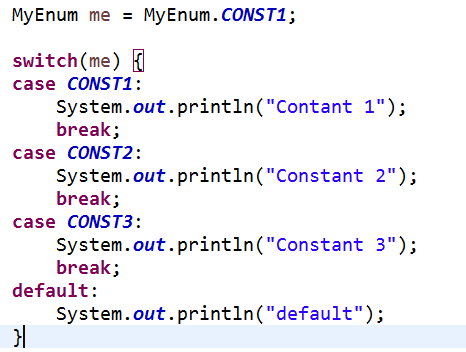
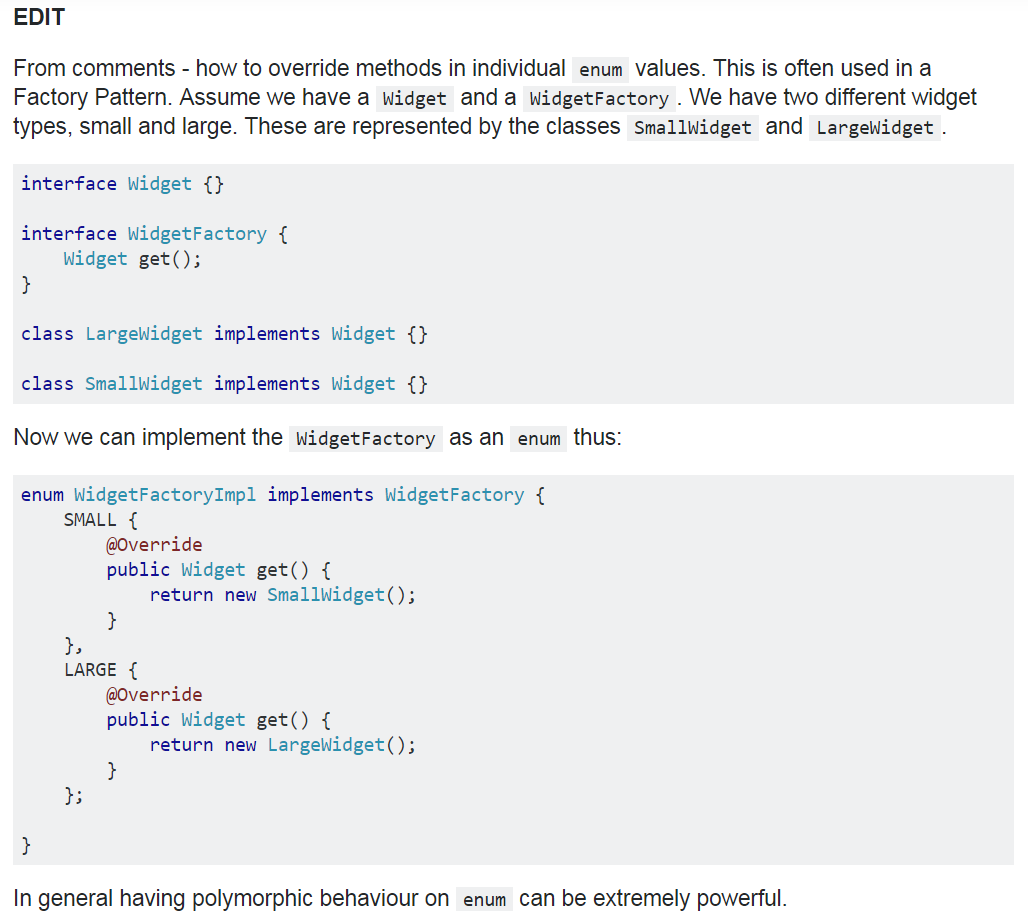
1. **Enumerations**: Enumeration represents class type for representing named constants of the enum class type.
   1. Enum constants are implicitly public, static and final
   2. We can create a variable of enum type but can’t initiate them using ‘new’



* 1. Enum can be used in a switch statement.



* 1. Enum can only be public or default since it can be inherited.
  2. An enum cannot extends another class because an enum already extends Enum<T>. That class provides all the enum functionality.
  3. An enum can, however, implements an interface.



* 1. Methods: ps enum-type [] values(), ps enum-type valueOf(String name),
     1. f int ordinal (return position of enum constant)
     2. f int compareTo(enum-type e) [compares ordinal]
     3. equals() [compare to any other object, returns true if object is of same type and ordinal]

1. **Type Wrappers**: Type wrapper represents object type for primitives e.g., Double, Float, Long, Integer, Short, Byte, Character, Boolean
   1. Numeric type wrappers inherit inherits abstract Number class.
   2. **Autoboxing**: Process of automatic conversion from primitive to wrapped type.
   3. **Auto-unboxing**: Process of automatic conversion from wrapped type to primitive type.
2. **Annotations**: Annotation enables addition of supplemental information into source file which doesn’t change the action of the program.

@Target({ ElementType.***METHOD***, ElementType.***FIELD*** })

@Retention(RetentionPolicy.***RUNTIME***)

**@interface** MyAnno {

String str();

**int** val();

**int** count() **default** 11;

}

@MyAnno(str = "main", val = 0)

**public** **static** **void** main(String... args) {

* 1. All annotation consists of method declaration and java implements them.
  2. Members can’t have parameters.
  3. Member can return.
  4. Annotation cannot include ‘extends’ clause but all annotations automatically extends ‘Annotation’ interface.
  5. Annotation type declaration can’t be generic.
  6. Fields, parameters, class, method, enum can be annotated.
  7. Retention policy: It determines at which point annotation is to be discarded.
     1. @Retention(RetentionPolicy.[SOURCE|CLASS|RUNTIME])
     2. CLASS is default retention policy.
  8. Getting annotation details:

Class<?> annoTest = AnnoTest.**class**; // use class literal when class is known

**try** {

Method m = annoTest.getMethod("main", String[].**class**);

**for** (Annotation a : m.getAnnotations()) {

System.***out***.println(a);

}

System.***out***.println(m.getAnnotation(MyMarker.**class**));

* 1. **AnnotatedElement**: This interface is implemented by Field, Method, Class, Constructor, package etc.
     1. Meyhods: getAnnotation(annoType), getAnnotations(), getDeclaredAnnoatations(), isAnnotationPresent(annoType)
        1. JDK 8: getDeclaredAnnotation(), getAnnotationByType(), getDeclaredAnnotationByType()
  2. **Default values**: Annotation members can be given default values.

**@interface** MyAnno {

String str();

**int** val();

**int** count() **default** 11;

}

* 1. **Marker annotations**: Annotation with no member.
  2. **Built-in annotations**:
     1. Java.lang.annotation: @Retention, @Documented, @Target, @Inherited
        1. @Target: @Target({ ElementType.***METHOD***, ElementType.***FIELD*** })
        2. ElementType: Constants defined in this enum are: ANNOTATION\_TYPE, CONSTRUCTOR, FIELD, LOCAL\_VARIABLE, METHOD, PACKAGE, PARAMETER, TYPE, TYPE\_PARAMETER, TYPE\_USE
     2. Java.lang: @Override, @Deprecated, @FunctionalInterface, @SafeVarags, @SuppressWarnings
  3. Type annotations: From JDK 8 annotations can be applied in cases where type in use e.g., this, array levels, an inherted class, throws clause, generic type and bound etc.

@Target(ElementType.***TYPE\_USE***)

**@interface** MyTypeAnno {

}

* + 1. They enable tools to perform additional check can be done by compiler.
    2. Usage:
       1. public class TypeAnno<@MyTypeAnno T>

@Target(ElementType.***TYPE\_USE***)

**@interface** MaxLen {

**int** value(); // this member name must be value otherwise must be specified while using the annotation

}

String @MaxLen(0) [] arr;

* + - 1. **public** **static** **void** main(String[] args) **throws** @MyTypeAnno NullPointerException
      2. **public** **void** m1(@MyTypeAnno TypeAnno<T> **this**) { // this "reciever" must be the first parameter

//allowed in java 8, to be used if need to annotate 'this'

}

* 1. **Repeating Annotation**: This annotation used to annotate an annotation that can be repeatedly used on the target.

@Repeatable(xyz.**class**)

**@interface** Schedule {

String dayOfMonth() **default** "first";

String dayOfWeek() **default** "Mon";

**int** hour() **default** 12;

}

**@interface** xyz {

Schedule[] value();

}

@Schedule(hour = 12)

@Schedule(dayOfMonth = "fri")

@SourceAnno("Source") // this shorthand works only if member name is 'value'

**public** **static** **void** main(String... args