**Abstraction in OOP**

* Abstraction means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones relevant to the project we develop
* Abstraction helps managing complexity
* Abstraction lets you focus on what the object does instead of how it does it.

**Interface**

* Internal addition by compiler

public interface Printable {

int MIN = 5;

void print();

}

Compiler

interface Printable {

public static final int MIN = 5;

public abstract void print();

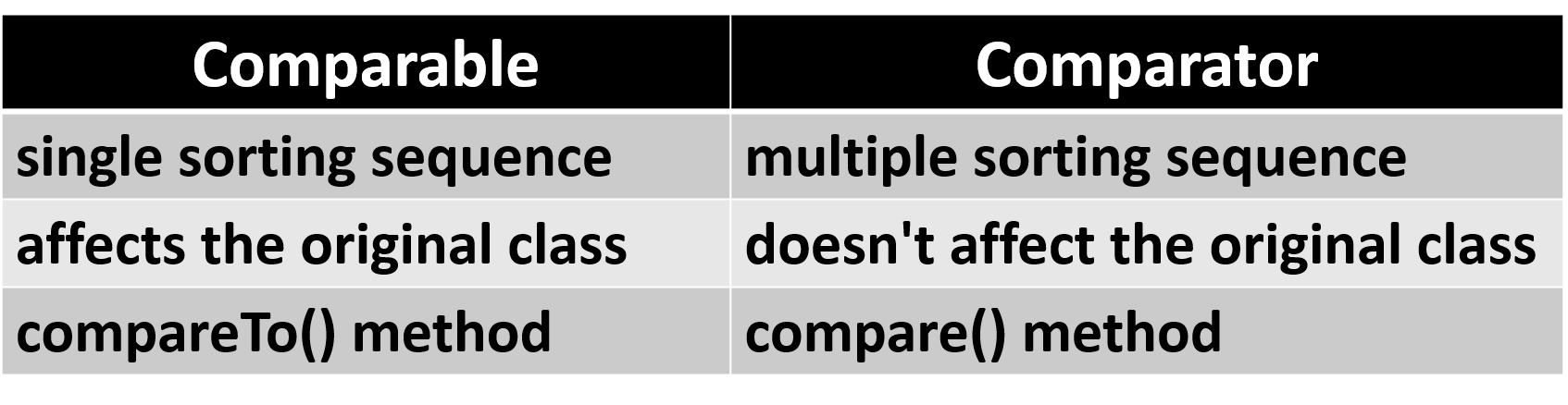
}

**Iterator<E> vs Enumeration<E>**

* Both Iterator and Enumeration allows you to traverse over elements of Collections
* Iterator takes the place of Enumeration in the Java collections framework.
  + - (hasNext(), next(), remove() );

**Comparator<E> vs Comparable<E>**

* Comparator provides a way for you to provide custom comparison logic for types that you have no control over.
* Comparable allows you to specify how objects that you are implementing get compared



**Generics Classes**

* Type variable T, that can be used anywhere inside the class

public class Box<T> {

// T stands for "Type"

private T t;

public void set(T t) { this.t = t; }

public T get() { return t; }

}

* This same technique can be applied to create generic interfaces.

**Generics Method**

* Type variable T, that can be used anywhere inside the method

public static <E> void printArray(E[] inputArray) {

// Display array elements

for (E element : inputArray) {

System.out.printf("%s ", element);

}

System.out.println();

}

* T can be returned type too

**Enums**

* Special data type that enables for a variable to be a set of predefined constants

public enum Level {

HIGH,

MEDIUM,

LOW

}

Level level = Level.MEDIUM

switch (level) {

case HIGH : ...; break;

case MEDIUM : ...; break;

case LOW : ...; break;

}

* This same technique can be applied to create generic interfaces.

**Annotations**

* Form of metadata, provide data about a program that is not part of the program itself
  + Information for the compiler
  + Compile-time and deployment-time processing
  + Runtime processing

@Override

void mySuperMethod() { ... }

@interface ClassPreamble {

String author();

String date();

int curRevision() default 1;

String[] reviewers();

}

**Reflection**

* Can inspect Java classes at runtime
* Inspecting classes is often the first thing you do when using Reflection

**Class myObjectClass = MyObject.class**

* You can get class name in two versions

String className = aClass.getName();

String simpleClassName = aClass.getSimpleName();

* You can reach almost everything in class

int modifiers = aClass.getModifiers();

Package package = aClass.getPackage();

Class superclass = aClass.getSuperclass();

Class[] interfaces = aClass.getInterfaces();

Constructor[] constructors = aClass.getConstructors();

Method[] method = aClass.getMethods();

Field[] method = aClass.getFields();

Annotation[] annotations = aClass.getAnnotations();