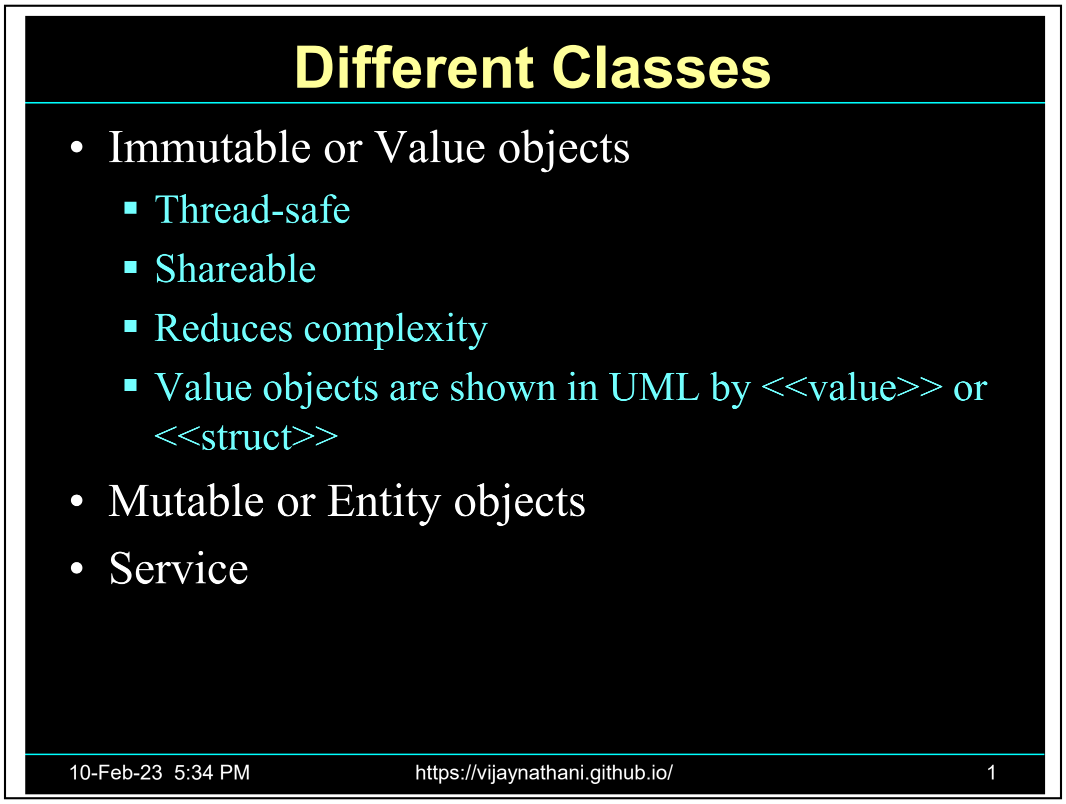


Different Classes

- Immutable or Value objects
  - Thread-safe
  - Shareable
  - Reduces complexity
  - Value objects are shown in UML by <<value>> or <<struct>>
- Mutable or Entity objects
- Service

10-Feb-23 5:34 PM https://vijaynathani.github.io/ 1

Q08 – Participant. Example of immutable using JUnit.

Service – Stateless.

Mutable. Keep state-space small, well-defined. Make clear when its legal to call which method.

Reference objects (Entities) and Value objects.

> Keep classes immutable unless there is a good reason to do otherwise.
>
> Enforcing contracts is a great way to reduce complexity. Simplifies development effort
>
> All instance variables should be “private final”.
>
> Value objects are not in any specific tier. They should be useable from all tiers.

================================

Immutable Example:

> Color, Line Style, Name, OrderNumber, ZipCode
>
> Company class has company address, company fax, company name, company Telephone, etc.
>
> Integers with limitations - Percentage(0 to 100%), Quantity (>=0)
>
> Arguments used in service methods

Exceptions that are domain logical

JDK examples

Bad – Date, Calendar, Dimension

Good - TimerTask

Just as String does not belong to any tier.

For many applications makes sense to

Use immutable classes most of the time

Handle mutations only in a part of the application

This limits complexity to one part

Rest of the application uses immutable classes to reduce complexity and enhance understanding.

E.g. Historic information, read from the database, should be immutable.

Example: A credit card engine

It needs to read details about Customer, Card and Transaction.

In the scope of this engine, all these classes are immutable.

Getters should normally return immutable objects.

java.util.collections.unmodifiableList

System.Collections.Generic. IEnumerable<T> or List<T>.AsReadOnly

----

Entities have an identity. So two account entities with same balance are not the same. They are almost always persistent.

Values are usually a part of Entity.

DTO (Data Transfer Objects) are different from Value objects. DTO is a

-purpose: data transfer – technical construct

- bunch of data – not necessarily coherent

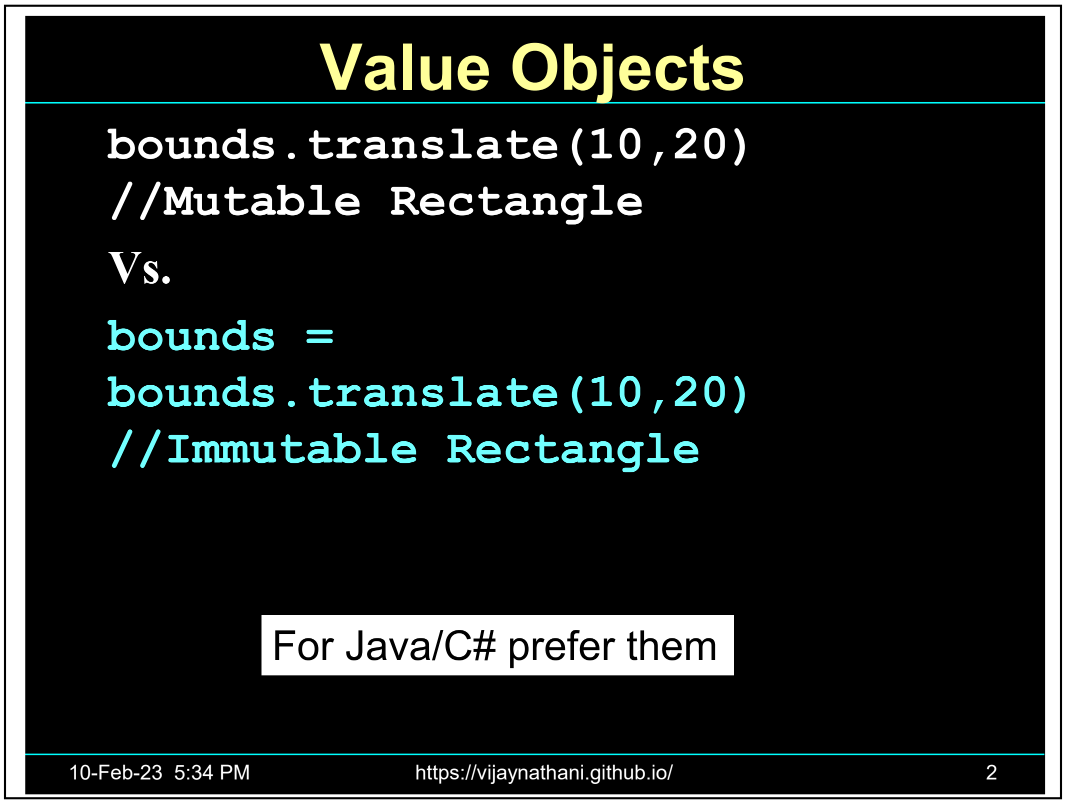
- no / little behavior

Value Object

-Purpose: domain representation

-High coherent data

-Rich on behavior



Value Objects

```
bounds.translate(10,20)
//Mutable Rectangle
```

Vs.

```
bounds =
bounds.translate(10,20)
//Immutable Rectangle
```

For Java/C# prefer them

10-Feb-23 5:34 PM https://vijaynathani.github.io/ 2

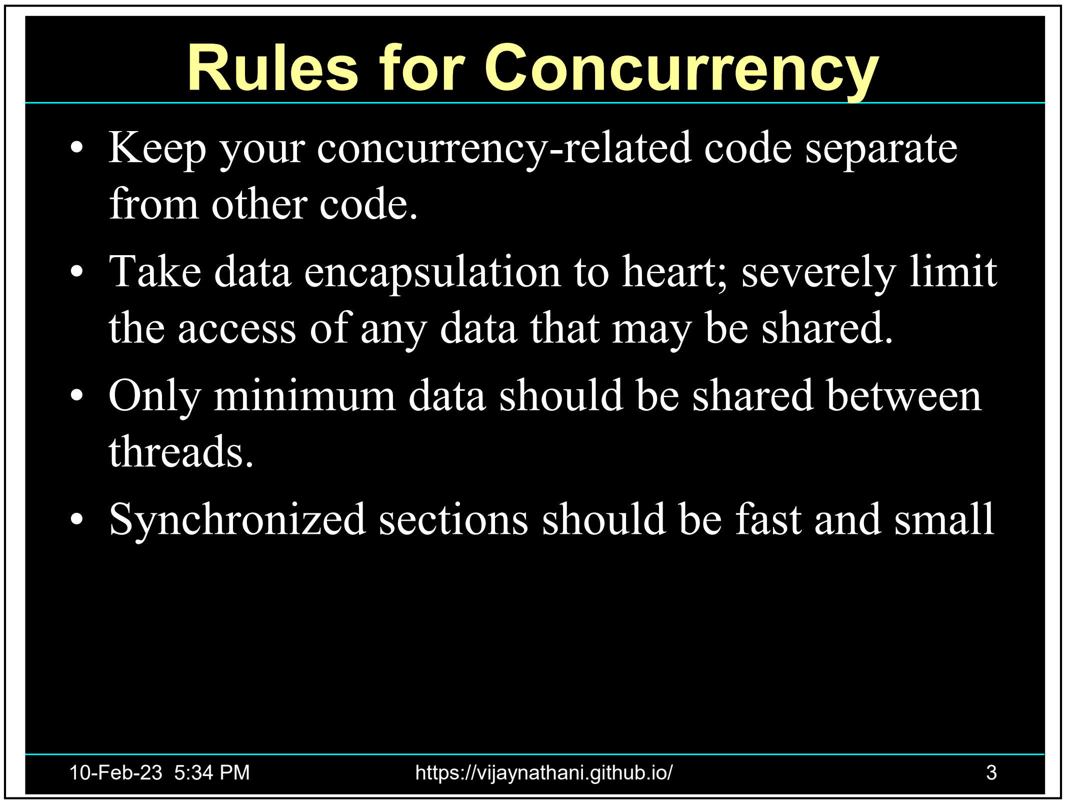
In modern functional languages, by default a variable is immutable i.e. its value cannot change

e.g. in F#

```
let x = 1 //the value of x cannot be changed.
let mutable y = 1 // the value of y can change.
```

In Scala

```
val x = 1; //x is immutable
var y = 1; //y is mutable
```

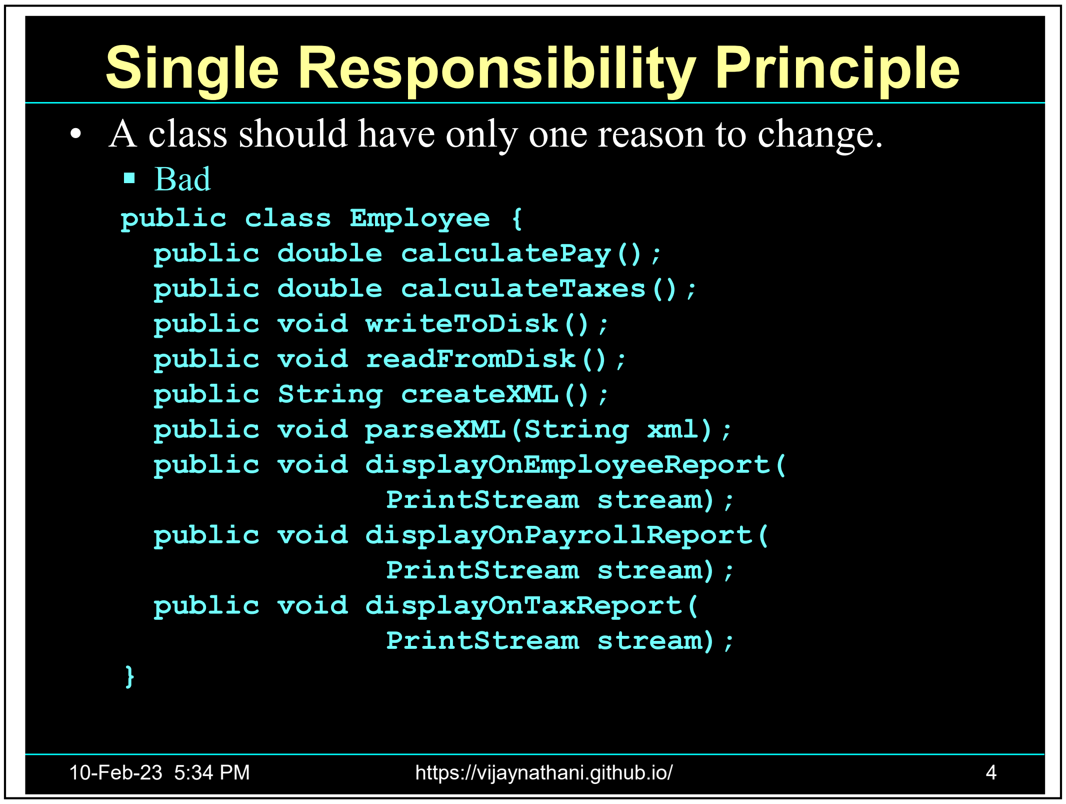


Rules for Concurrency

- Keep your concurrency-related code separate from other code.
- Take data encapsulation to heart; severely limit the access of any data that may be shared.
- Only minimum data should be shared between threads.
- Synchronized sections should be fast and small

10-Feb-23 5:34 PM https://vijaynathani.github.io/ 3

Return copies of data or use immutable objects.



Single Responsibility Principle

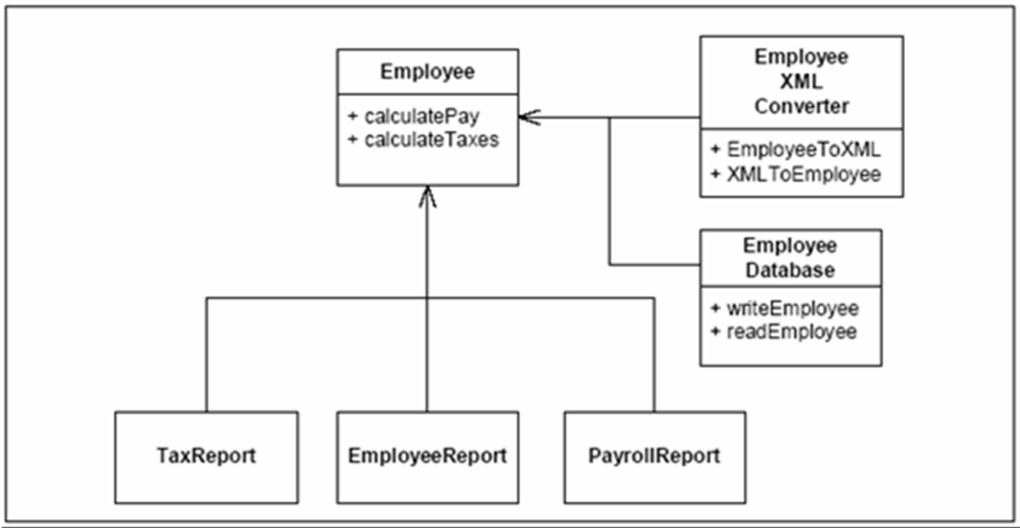
- A class should have only one reason to change.
  - Bad

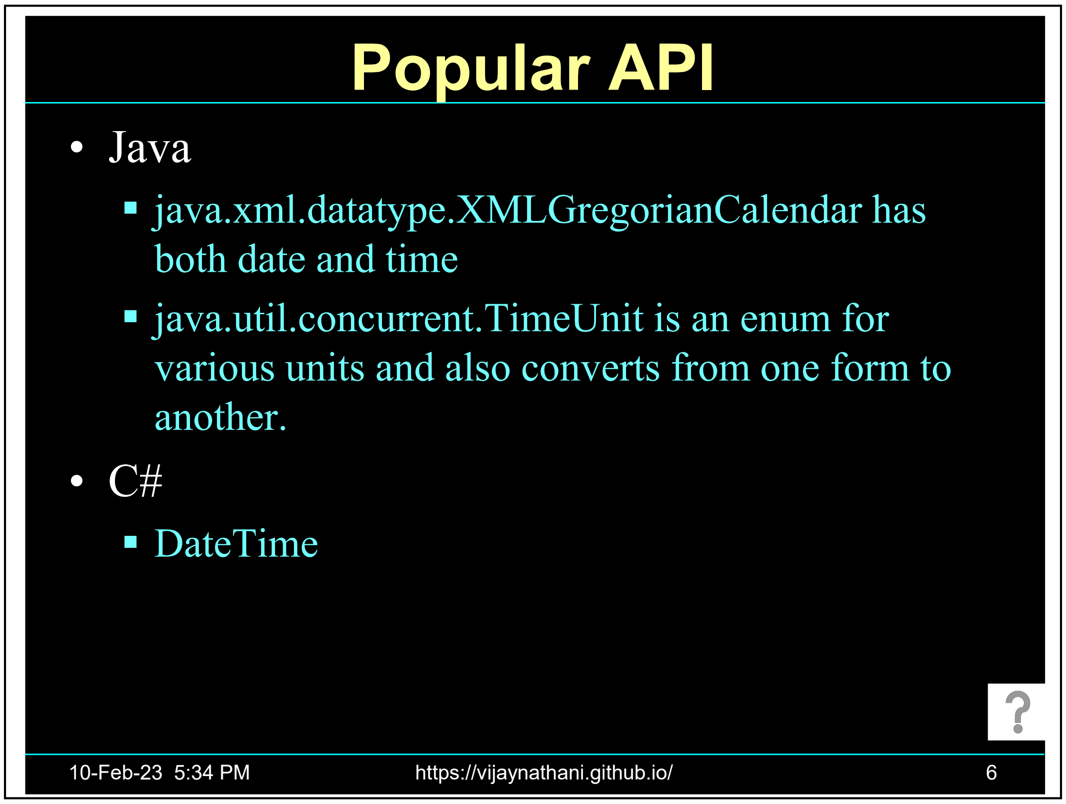
```
public class Employee {
  public double calculatePay();
  public double calculateTaxes();
  public void writeToDisk();
  public void readFromDisk();
  public String createXML();
  public void parseXML(String xml);
  public void displayOnEmployeeReport(
                 PrintStream stream);
  public void displayOnPayrollReport(
                 PrintStream stream);
  public void displayOnTaxReport(
                 PrintStream stream);
}
```

10-Feb-23 5:34 PM https://vijaynathani.github.io/ 4

An item such as a class should just have one responsibility and solve that responsibility well. If a class is responsible both for presentation and data access, that's a good example of a class breaking SRP.

SRP implemented





Popular API

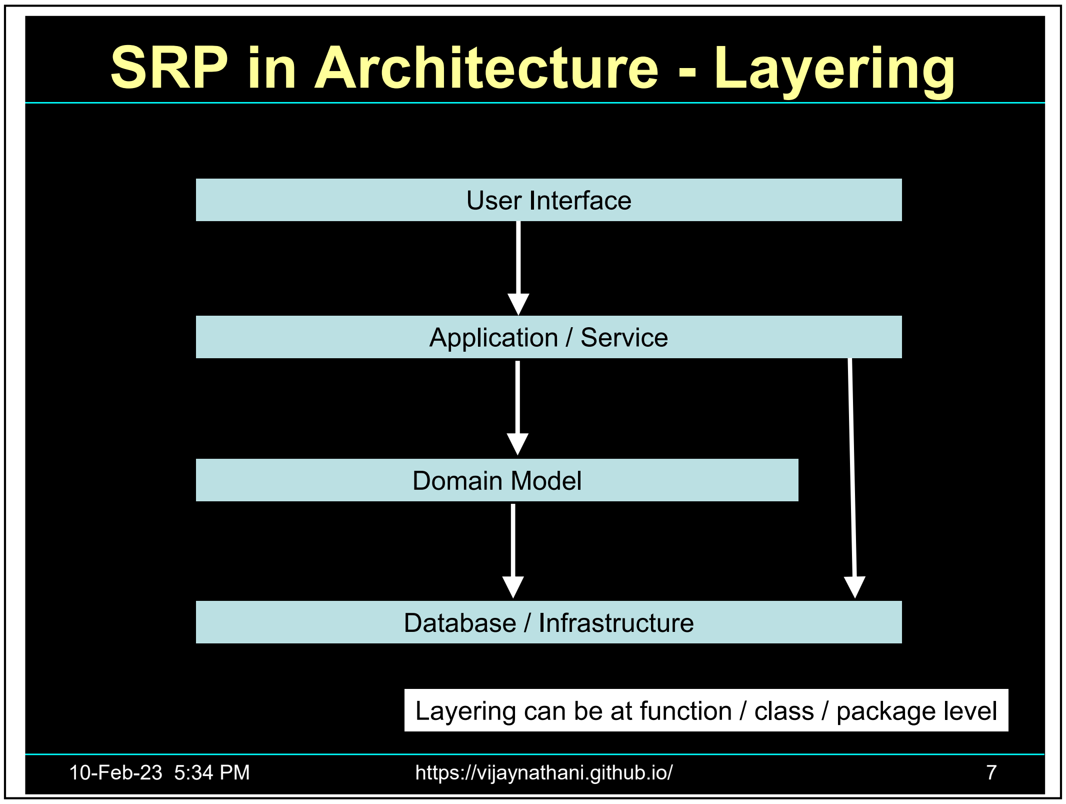
- Java
  - java.xml.datatype.XMLGregorianCalendar has both date and time
  - java.util.concurrent.TimeUnit is an enum for various units and also converts from one form to another.
- C#
  - DateTime

10-Feb-23 5:34 PM https://vijaynathani.github.io/ 6

Q50 – Account

Q51 – Department

Q55 – processReport1



SRP in Architecture - Layering

Layering can be at function / class / package level

10-Feb-23 5:34 PM https://vijaynathani.github.io/ 7

Service / Application layer co-ordinates tasks and delegates work to the domain. In an rich domain model, Service layer should be as thin as possible.

Interactions with the legacy systems is at Infrastructure level.

Usually the service layer

1. Gets the Database objects by interacting directly with Database layer and then
2. The domain model is executed in the objects.

In Java EE 1.4 or less,

Session beans are at service layer

Entity beans are at Database layer.

If business logic code is kept in Session beans, then it will become transaction script.

If business logic code is kept in Entity beans, then it will become Table Module

Table module is popular in .NET. Transaction script was popular with VB6.

One of the hardest parts of working with domain logic seems to be that people often find it difficult to recognize what is domain logic and what is other forms of logic. An informal test I like is to imagine adding a radically different layer to an application, such as a command-line interface to a Web application. If there's any functionality you have to duplicate in order to do this, that's a sign of where domain logic has leaked into the presentation. Similarly, do you have to duplicate logic to replace a relational database with an XML file?