# Design Principles

**SOLID Principles (Class Design Principles)**

* [**Single Responsibility Principle**](http://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#SRP) : One class should have one and only one responsibility.
* [**Open Closed Principle**](http://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#OCP): Software components should be open for extension, but closed for modification.
* [**Liskov's Substitution Principle**](http://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#LSP): Derived type must be completely substitutable for their base types.
* [**Interface Segregation Principle**](http://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#ISP): Clients should not be forced to implement unnecessary methods which they will not use.
* [**Dependency Inversion Principle**](http://howtodoinjava.com/best-practices/5-class-design-principles-solid-in-java/#DI) **:** Depend on abstraction and not on concretions.

# Writing Junit Test Class for an Existing Class

**Test Driven Development (TDD):**

What is TDD:

1. You read and understand requirements for a particular feature.
2. You develop set of tests which check the feature. All of the tests are red, due to absence of the feature implementation.
3. You develop the feature until all tests become green.
4. Refactoring of the code.

That is write unit test class first than the implementation class ( or TDD is a development of tests before a feature implementation).

A Unit test must be:

* Test behaviour but not methods
* Be small and fast
* Have good name
* Be well grained
* Be predictable

**FIRST Principles of Good Unit Tests**

Acronym FIRST stand for below test features:

* [F]ast
* [I]solated
* [R]epeatable
* [S]elf-validating
* [T]imely

**Behaviour Driven Development(BDD):**

Behaviour-driven development is an extension of [test-driven development](https://en.wikipedia.org/wiki/Test-driven_development):

BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters.

BDD focused on:

* Where to start in the process
* What to test and what not to test
* How much to test in one go
* What to call the tests
* How to understand why a test fails

**A simplified Waterfall model would be:**

1. Requirements specification resulting in requirements document
2. Design resulting in software architecture document
3. Development resulting in actual software
4. Integration
5. Testing
6. Installation
7. Maintenance

Each of these phases tends to have separate teams and departments. BAs work with requirements, architects write design documents, developers code, integration engineers integrate, testers test, and someone installs the software.

## **Authentication**

## **Authorization**

## **Session Management**

## **Caching Mechanism**

## **Connection Pooling**

## **Database Designing**