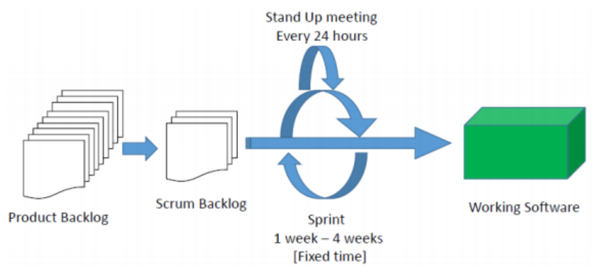
The **File** class

- Contains methods for obtaining the properties of a file/directory and for renaming and deleting a file/directory
- Files could be specified using absolute or relative names
- Constructing a **File** instance does not create a file on the machine
- Methods include:
  - ➢ **boolean createNewFile()**
  - ➢ **boolean delete()**
  - ➢ **boolean exists()**
  - ➢ **boolean isDirectory()**
  - ➢ **File [] listFiles()**

- Reading could be done using the **Scanner** class
  - ➢ E.g. **Scanner input = new Scanner(new File(filename));**
- Writing could be done using the **FileWriter** class
  - ➢ E.g. **FileWriter output = new FileWriter(filename, append);**

Regular Expression	*Matches*	*Example*
`.`	any single character	`Java matches J..a`
`(ab|cd)`	ab or cd	`ten matches t(en|im)`
`[abc]`	a, b, or c	`Java matches Ja[uvwx]a`
`[^abc]`	any character except a, b, or c	`Java matches Ja[^ars]a`
`[a-z]`	a through z	`Java matches [A-M]av[a-d]`
`[^a-z]`	any character except a through z	`Java matches Jav[^b-d]`
`[a-e[m-p]]`	a through e or m through p	`Java matches [A-G[I-M]]av[a-d]`
`[a-e&&[c-p]]`	intersection of a-e with c-p	`Java matches [A-P&&[I-M]]av[a-d]`
`\d`	a digit, same as [0-9]	`Java2 matches "Java[\\d]"`
`\D`	a non-digit	`$Java matches "[\\D][\\D]ava"`
`\w`	a word character	`Java1 matches "[\\w]ava[\\w]"`
`\W`	a non-word character	`$Java matches "[\\W][\\w]ava"`
`\s`	a whitespace character	`"Java 2" matches "Java\\s2"`
`\S`	a non-whitespace char	`Java matches "[\\S]ava"`
*p**	zero or more occurrences of pattern *p*	`aaaabb matches "a*bb"` `ababab matches "(ab)*"`
p+	one or more occurrences of pattern *p*	`a matches "a+b*"` `able matches "(ab)+.*"`
p?	zero or one occurrence of pattern *p*	`Java matches "J?Java"` `Java matches "J?ava"`
p{n}	exactly n occurrences of pattern *p*	`Java matches "Ja{1}.*"` `Java does not match ".{2}"`
p{n,}	at least n occurrences of pattern *p*	`aaaa matches "a{1,}"` `a does not match "a{2,}"`
p{n,m}	between n and m occurrences (inclusive)	`aaaa matches "a{1,9}"` `abb does not match "a{2,9}bb"`

- Scrum is currently one of the most widely used methodologies of software development



- Product Owner
  - ➢ Responsible for delivering requirements and accepting demos
  - ➢ Involved in planning session
- Scrum Master
  - ➢ Responsible for removing impediments
- Team members
  - ➢ No one has a fixed role other than the scrum master and product owner
  - ➢ Everyone takes on tasks, and completes them based on what they are most comfortable with

- Happens EVERY SINGLE day that you are working
- The goal is to make sure people are doing alright
- Shouldn't be longer than 15 minutes
- Answer three questions:
  1. What did I finish since the last standup?
  2. What am I going to finish by the next standup?
  3. What is stopping me / what impediments am I facing?

Regular Expressions

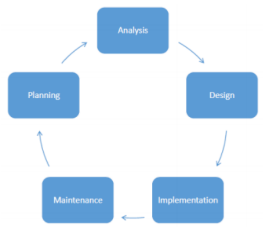
- Java supports regular expressions using the **java.util.regex** package
- The **Pattern** class can be used to define the pattern
  - ➢ The **compile** method takes a string representing the regular expression as an argument and compiles it into a pattern
- The **Matcher** class can be used to search for the pattern. Its methods include:
  - ➢ **boolean find()**
  - ➢ **boolean matches()**
- Example

```
Pattern pattern = Pattern.compile("H.*d");
Matcher matcher = pattern.matcher("Hello World");
System.out.printl(matcher.matches());
```

- **System.in**
  - ➢ Object of type **InputStream**
  - ➢ Typically refers to the keyboard
  - ➢ Reading data could be done using the **Scanner** class. Its methods include:
    - o **String next()**
    - o **String nextLine()**
    - o **int nextInt()**
    - o **double nextDouble()**
- **System.out**
  - ➢ Object of type **PrintStream**
  - ➢ Typically refers to the console

Software Development Life Cycle (SDLC)

- **Planning**–develop a plan for creating the concept or evolution of the concept
- **Analysis**–analyze the needs of those using the system. Create detailed requirements
- **Design**–Translate the detailed requirements into detailed design work
- **Implementation**–Complete the work of developing and testing the system
- **Maintenance**–Complete any required maintenance to keep the system running



	Agile	Waterfall
Iterative?	Yes	No
Late Changes?	Yes	No / $$$
Fixed timeline?	No*	Yes
Fixed Cost?	No*	Yes*
Volume of meetings	Consistent	Heavy up front, reduced middle, heavy end
Release frequency	Every Sprint	Once per project
Business Involvement	Heavy throughout	Heavy early, and at very end
Cost to fix mistakes	Low	High

eXtreme Programming (XP)

- One of the most rigorous forms of Agile
- Involves building a series of feedback loops, which are used to help guide when change can occur and allow for changes to be quickly integrated into the plan for development
- Built on the idea that you can reduce the cost of developing software, and build better software, by having goals
- XP requires that everything that can be unit tested is unit tested, that everyone works in pairs, and that these pairs change frequently.

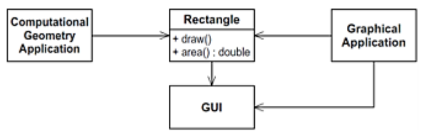
- The sprint is a fixed time to deliver a working set of features, that are reviewed in a demonstration to the product owner
- Tasks in Scrum are broken into "User Stories"
- In a sprint, a team agrees at the beginning to take on a certain number of user stories
- Sprints are usually between 1 and 4 weeks in length
- At the end of each sprint, teams hold a "retrospective" which is a meeting where the past sprint is discussed, and chances for improvement for the next sprint are raised

As a {ACTOR/OBJECT} I want to {ACTION} so that {RESULT}

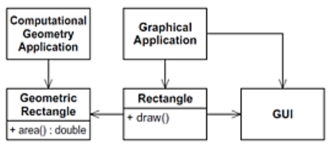
- In scrum, we do not assign time to tasks, but assign arbitrary points. This is a form of estimation that helps gauge how much work something will take to complete.
- Planning poker takes a set of pre-determined numbers (usually: 1, 2, 3, 5, 8, etc.) and gets you to estimate how much work something will be relative to a known task.
- After discussing the story at hand , everyone selects a card. Then, the cards are turned over simultaneously. Usually time is given for those who had the lowest and highest numbers to state their case
- The process is repeated until everyone ends up at the same number.

- Planning sessions happen at the start of each sprint.
- They usually take a few hours. During this time, the team decides how much work it will take on, and discusses any major technical challenges they expect to face.
- Usually, Product Owners are available for at least a portion of this meeting, to help with prioritization. They are only there to assist in this regard, and not to dictate what the team will complete.

Violating the SRP (example)

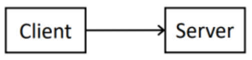


Conforming to the SRP (example)



Violating the OCP (example)

- Both classes are concrete
- The **Client** uses the **Server** class

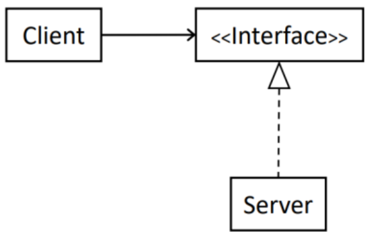


The Open/Closed Principle (OCP)

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

- When a single change to a program results in a cascade of changes to dependent modules, the design smells of rigidity.
  - ➢ If the Open/Closed principle is applied well, then further changes of that kind are achieved by adding new code, not by changing old code that already works.
- In Java, it is possible to create abstractions that are fixed and yet represent an unbounded group of possible behaviors
  - ➢ The abstractions are abstract base classes, and the unbounded group of possible behaviors is represented by all the possible derivative classes.

Conforming to the OCP (example)



The Liskov Substitution Principle (LSP)

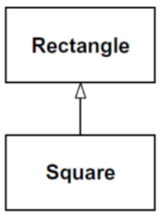
Subtypes must be substitutable for their base types.

- Formally: *Let Φ(x) be a property provable about objects x of type T. Then Φ(y) should be true for objects y of type S where S is a subtype of T.*
- Counter-example: *"If it looks like a duck, quacks like a duck, but needs batteries – you probably have the wrong abstraction"*

Violating the LSP (example)

Issues
- Inheriting **height** and **width**
- Overriding **setHeight** and **setWidth**
- Conflicting assumptions. For example:

```
void testRectangleArea(Rectangle r){
        r.setWidth(5);
        r.setHeight(4);
        assertEquals(r.computeArea(), 20);
}
```

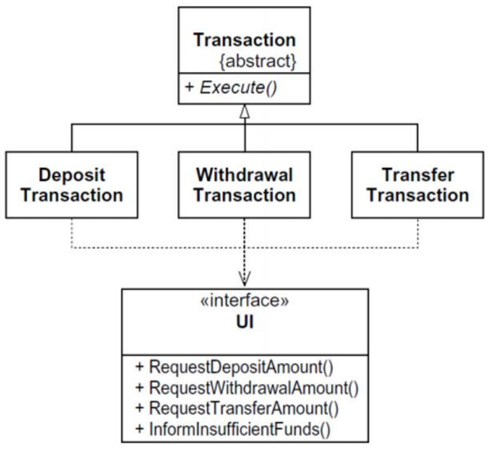


The Interface Segregation Principle (ISP)

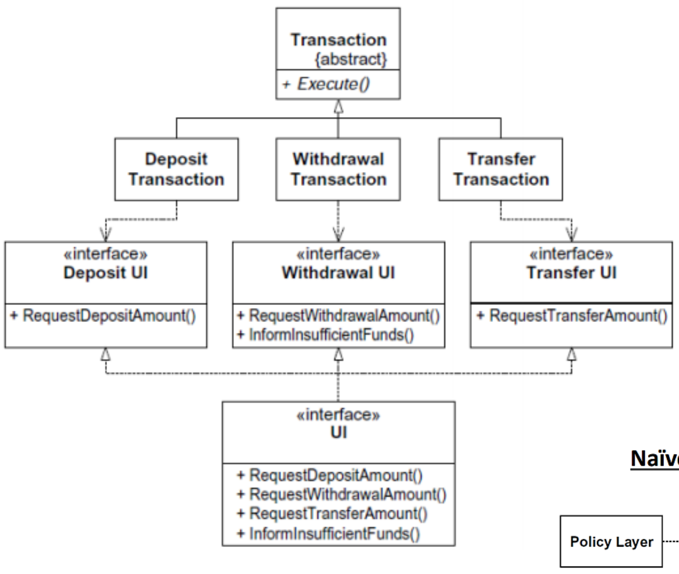
Clients should not be forced to depend on methods that they do not use.

- This principle deals with classes whose interfaces are not cohesive. That is, the interfaces of the class can be broken up into groups of methods where each group serves a different set of clients.
- When clients are forced to depend on methods they don't use, then those clients are subject to changes to those methods.

Violating the ISP (example)



Not violating ISP:



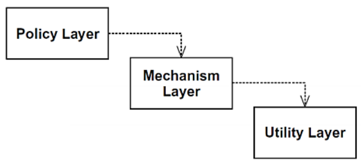
The Dependency-Inversion Principle (DIP)

A. High-level modules should not depend on low-level modules. Both should depend on abstractions.

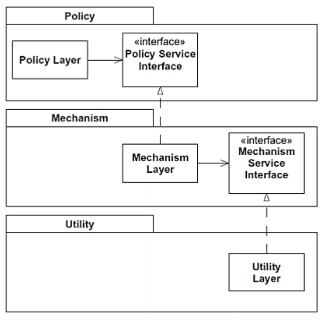
B. Abstractions should not depend on details. Details should depend on abstractions.

- The modules that contain the high-level business rules should take precedence over, and be independent of, the modules that contain the implementation details.
- When high-level modules depend on low-level modules, it becomes very difficult to reuse those high-level modules in different contexts.

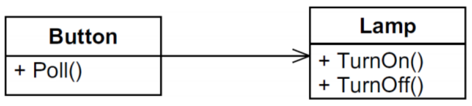
<u>Naïve Layering</u>



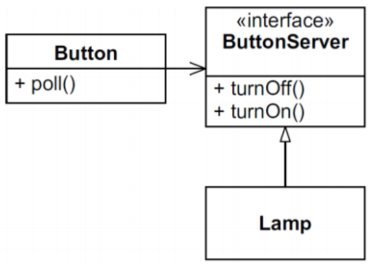
<u>Inverted Layers</u>



Violating the DIP (example)



Conforming to the DIP (example)



- Symptoms of poor design
- Often caused by the violation of one or more of the design principles
  - ➢ For example, the smell of *Rigidity* is often a result of insufficient attention to OCP.
- These symptoms include:
  1. Rigidity—The design is hard to change.
  2. Fragility—The design is easy to break.
  3. Immobility—The design is hard to reuse.
  4. Viscosity—It is hard to do the right thing.
  5. Needless Complexity—Overdesign.
  6. Needless Repetition—Mouse abuse.
  7. Opacity—Disorganized expression.

😊