Student: 16200403

Name: Bhargavi Patel

Overview:

The lecture started off with the principles of software engineering:

Correctness: The software needs to give the expected output

Robust: The software should prevent abnormal exits and handle them in any case.

loosely coupled: The submodule should be minimally dependent on the other modules.

data abstraction: Present the high level understandable unit to user and hide the implementation.

Maintainable: Can be maintained at fine grained level very easily and quickly. Code should not be tangled for this.

Testable : Should pass for all the expected input parameters.

Minimal: Code should be minimal and not repeated. Use of code reusability and modularity is important in this case.

Design patterns features are mainly used for 2 main purposes:

Then we moved on to discuss about the object oriented concepts:

abstraction: high level presentation and implementation difference

encapsulation: hiding implementation details

cohesive: self sufficient

coupling : independent or dependent on other modules for complete execution

SOLID principles: SOLID principles are a precursor to patterns and all the patters make use of these principles. They are:

Single responsibility principle

A class should have only a singularly responsibility

Open/closed principle

open for extension, but closed for modification

Liskov substitution principle

objects should be replaceable with instances of their subtypes without altering the correctness

Interface segregation principle

client-specific interfaces are better than one general-purpose interface

Dependency inversion principle

Depend upon Abstractions. Do not depend upon concretions

-

Interfaces can be coupled to classes based on 2 criterias:

services a class provides: printer(example taken in class) is providing the printing service. printing interface would be couples to printer in this case.

requirement of a class: client needs a printing service. In this case, the interface will look for a plugin that would fulfil this criteria and connect to it to service the client.

Disadvantages of implicit dependency : where modules are tightly coupled instantiating a class from another class creates a dependency that requires updating of both classes every time.

template pattern: see book osoles

unstable module can be converted instable one by subclassing the unstable part and calling the appropriate subclasses with change of parameter.

Importance of subclassing in template pattern :subclasses can be used to implement abstract methods

Downside: need to instantiate each subclass in one call to execute multiple requirements/formats

hook method: implementation in super class need not overwrite in subclass

These should have a naming convention so developer knows they are available for implementation as they are not mandatory

selection of pattern should be based on:

where what is expected

passing data between classes :

1. using parameters

2. using self(disadv is we need to state the context as public which we would have set as private.)

scope can be dynamically changed in ruby which helps using self for passing context

Strategy pattern:

-context class calls a strategy class which decides which best strategy to solve it

Key Points:

- interface is used to reduce coupling between classes
- Good practice: seperate out the unstable components (which needs to be changed)
- do not connect unstable bits with the stable ones
- strings are mutable in ruby and immutable in java so use symbols instead
- Only apply hook methods when the context demands it.
- Problem definition should not be defined in superclass. It is better to defer it to sub class