Design Defects and Restructuring

Engr. Abdul-Rahman Mahmood

abdulrahman@nu.edu.pk

alphapeeler.sf.net/pubkeys/pkey.htm

pk.linkedin.com/in/armahmood

www.twitter.com/alphapeeler

www.facebook.com/alphapeeler

abdulmahmood-sss alphasecure

armahmood786

http://alphapeeler.sf.net/me

alphapeeler#9321

reddit.com/user/alphapeeler

www.flickr.com/alphapeeler

http://alphapeeler.tumblr.com

armahmood786@jabber.org

alphapeeler@aim.com

mahmood_cubix 48660186

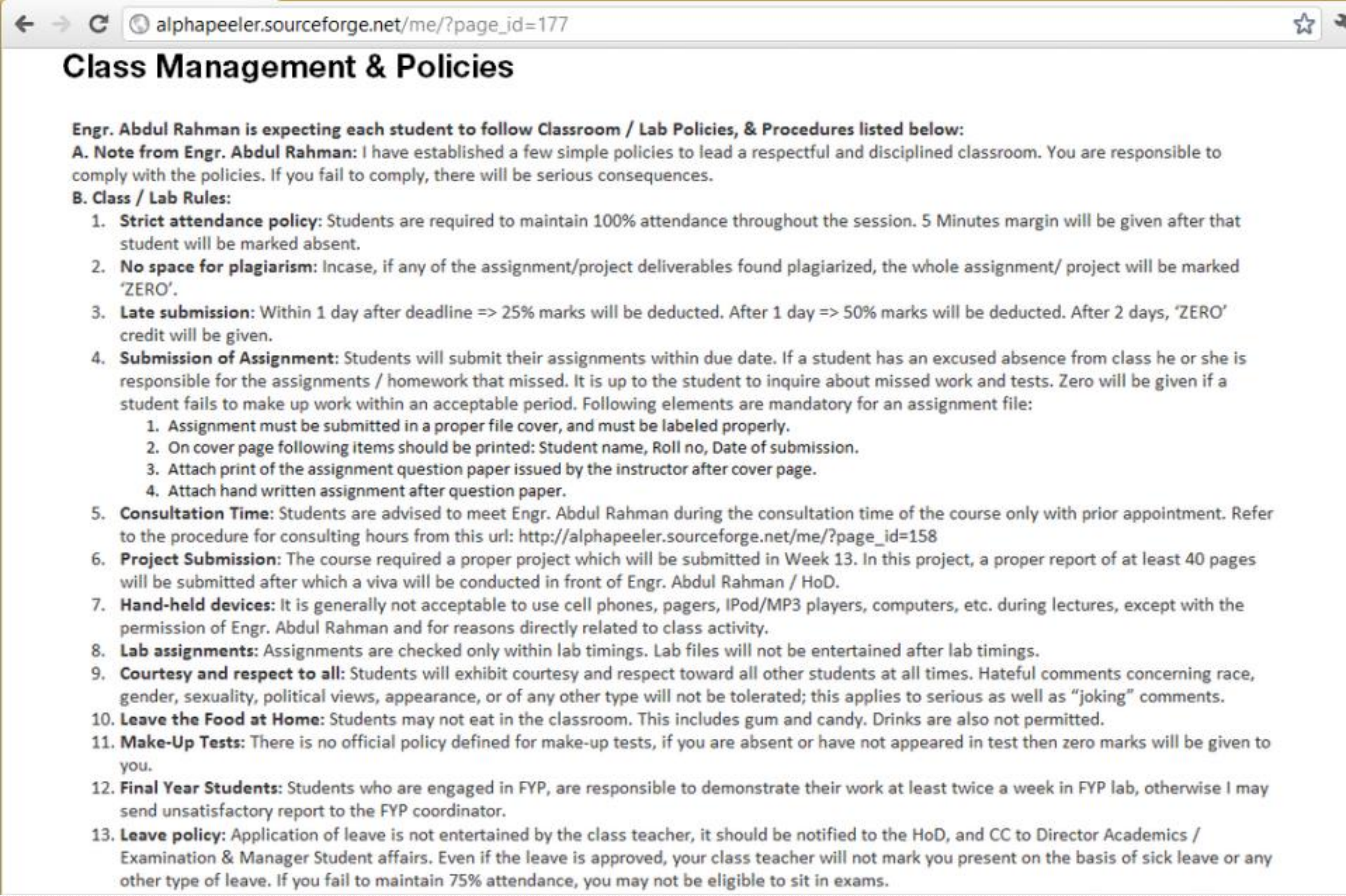
alphapeeler@icloud.com

pinterest.com/alphapeeler

www.youtube.com/user/AlphaPeeler

About The Course

Class Policies



alphapeeler.sourceforge.net/me/?page_id=177

Class Management & Policies

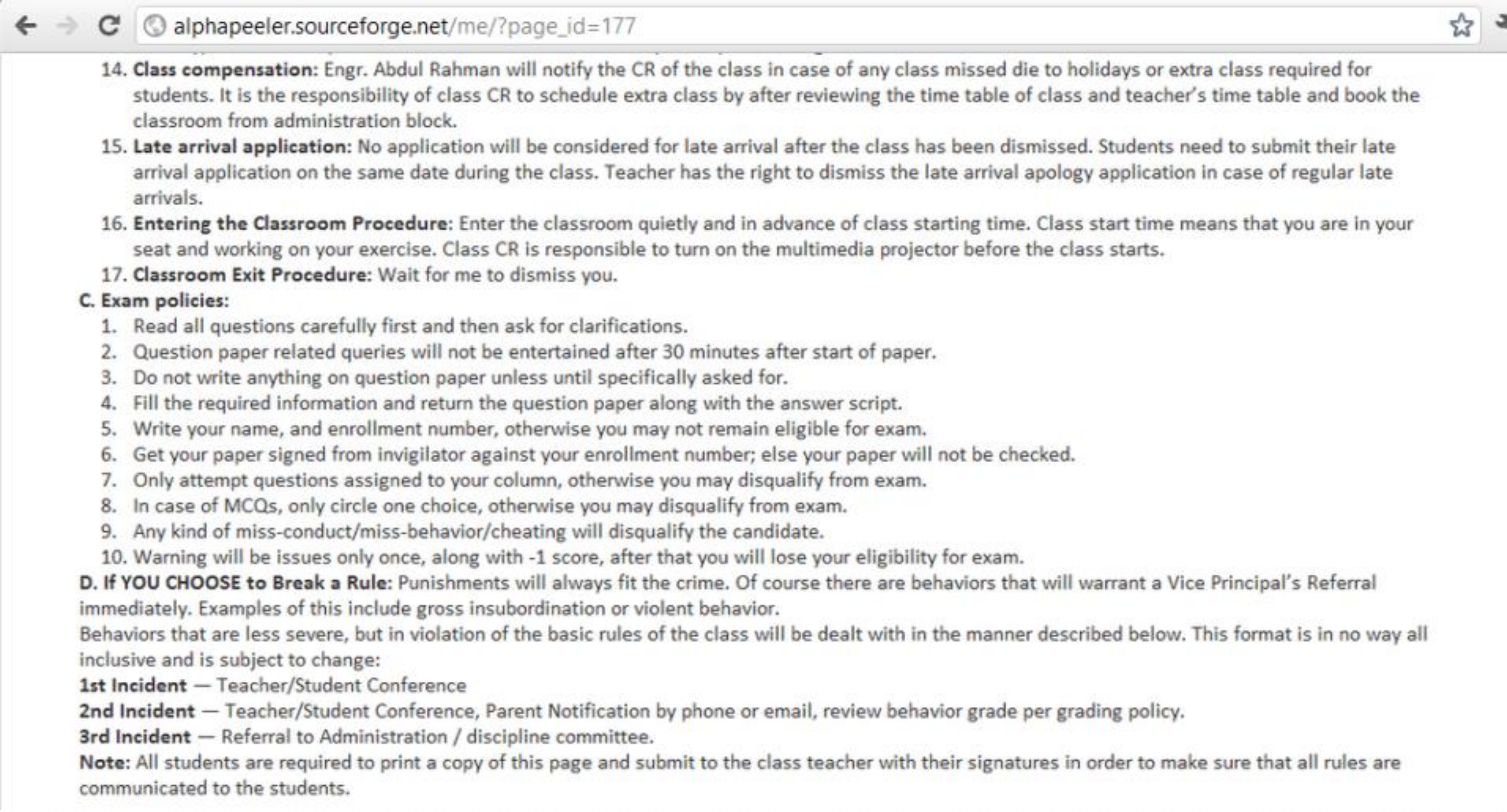
Engr. Abdul Rahman is expecting each student to follow Classroom / Lab Policies, & Procedures listed below:

A. Note from Engr. Abdul Rahman: I have established a few simple policies to lead a respectful and disciplined classroom. You are responsible to comply with the policies. If you fail to comply, there will be serious consequences.

B. Class / Lab Rules:

1. **Strict attendance policy:** Students are required to maintain 100% attendance throughout the session. 5 Minutes margin will be given after that student will be marked absent.
2. **No space for plagiarism:** Incase, if any of the assignment/project deliverables found plagiarized, the whole assignment/ project will be marked 'ZERO'.
3. **Late submission:** Within 1 day after deadline => 25% marks will be deducted. After 1 day => 50% marks will be deducted. After 2 days, 'ZERO' credit will be given.
4. **Submission of Assignment:** Students will submit their assignments within due date. If a student has an excused absence from class he or she is responsible for the assignments / homework that missed. It is up to the student to inquire about missed work and tests. Zero will be given if a student fails to make up work within an acceptable period. Following elements are mandatory for an assignment file:
   1. Assignment must be submitted in a proper file cover, and must be labeled properly.
   2. On cover page following items should be printed: Student name, Roll no, Date of submission.
   3. Attach print of the assignment question paper issued by the instructor after cover page.
   4. Attach hand written assignment after question paper.
5. **Consultation Time:** Students are advised to meet Engr. Abdul Rahman during the consultation time of the course only with prior appointment. Refer to the procedure for consulting hours from this url: http://alphapeeler.sourceforge.net/me/?page_id=158
6. **Project Submission:** The course required a proper project which will be submitted in Week 13. In this project, a proper report of at least 40 pages will be submitted after which a viva will be conducted in front of Engr. Abdul Rahman / HoD.
7. **Hand-held devices:** It is generally not acceptable to use cell phones, pagers, IPod/MP3 players, computers, etc. during lectures, except with the permission of Engr. Abdul Rahman and for reasons directly related to class activity.
8. **Lab assignments:** Assignments are checked only within lab timings. Lab files will not be entertained after lab timings.
9. **Courtesy and respect to all:** Students will exhibit courtesy and respect toward all other students at all times. Hateful comments concerning race, gender, sexuality, political views, appearance, or of any other type will not be tolerated; this applies to serious as well as "joking" comments.
10. **Leave the Food at Home:** Students may not eat in the classroom. This includes gum and candy. Drinks are also not permitted.
11. **Make-Up Tests:** There is no official policy defined for make-up tests, if you are absent or have not appeared in test then zero marks will be given to you.
12. **Final Year Students:** Students who are engaged in FYP, are responsible to demonstrate their work at least twice a week in FYP lab, otherwise I may send unsatisfactory report to the FYP coordinator.
13. **Leave policy:** Application of leave is not entertained by the class teacher, it should be notified to the HoD, and CC to Director Academics / Examination & Manager Student affairs. Even if the leave is approved, your class teacher will not mark you present on the basis of sick leave or any other type of leave. If you fail to maintain 75% attendance, you may not be eligible to sit in exams.

Class Policies


alphapeeler.sourceforge.net/me/?page_id=177

14. **Class compensation:** Engr. Abdul Rahman will notify the CR of the class in case of any class missed die to holidays or extra class required for students. It is the responsibility of class CR to schedule extra class by after reviewing the time table of class and teacher's time table and book the classroom from administration block.
15. **Late arrival application:** No application will be considered for late arrival after the class has been dismissed. Students need to submit their late arrival application on the same date during the class. Teacher has the right to dismiss the late arrival apology application in case of regular late arrivals.
16. **Entering the Classroom Procedure:** Enter the classroom quietly and in advance of class starting time. Class start time means that you are in your seat and working on your exercise. Class CR is responsible to turn on the multimedia projector before the class starts.
17. **Classroom Exit Procedure:** Wait for me to dismiss you.

C. Exam policies:

1. Read all questions carefully first and then ask for clarifications.
2. Question paper related queries will not be entertained after 30 minutes after start of paper.
3. Do not write anything on question paper unless until specifically asked for.
4. Fill the required information and return the question paper along with the answer script.
5. Write your name, and enrollment number, otherwise you may not remain eligible for exam.
6. Get your paper signed from invigilator against your enrollment number; else your paper will not be checked.
7. Only attempt questions assigned to your column, otherwise you may disqualify from exam.
8. In case of MCQs, only circle one choice, otherwise you may disqualify from exam.
9. Any kind of miss-conduct/miss-behavior/cheating will disqualify the candidate.
10. Warning will be issues only once, along with -1 score, after that you will lose your eligibility for exam.

D. If YOU CHOOSE to Break a Rule: Punishments will always fit the crime. Of course there are behaviors that will warrant a Vice Principal's Referral immediately. Examples of this include gross insubordination or violent behavior.

Behaviors that are less severe, but in violation of the basic rules of the class will be dealt with in the manner described below. This format is in no way all inclusive and is subject to change:

1st Incident — Teacher/Student Conference

2nd Incident — Teacher/Student Conference, Parent Notification by phone or email, review behavior grade per grading policy.

3rd Incident — Referral to Administration / discipline committee.

Note: All students are required to print a copy of this page and submit to the class teacher with their signatures in order to make sure that all rules are communicated to the students.

Consultation - INSTRUCTIONS

1. Instructor **will not entertain** any FYP & Project consultations **in the week before mid1, mid2 and final exams**. Consultations will resume after mid1, and mid2 exams as usual.
2. First step for consultation meeting is to choose the suitable time slot from above schedule, and send me an email for consultation request on my official email address: abdulrahman@nu.edu.pk at least three days earlier so that I can book the slot for you in my calendar. If I am busy in your requested time slot, I will respond you with an alternate slot.
3. If I have confirmed you the meeting time slot, your scheduled meeting will appear in my Google calendar on this URL: https://goo.gl/Wj2GM7
4. Please book meetings by Google's standard meeting request procedure: https://goo.gl/qixbaj
5. Bring the consultation form along with you when you meet me. Consultation Request Form from can be downloaded from here.
6. For FYP students, it is mandatory to consult with me once in a week to report the project status.
7. Your project status will appear on the project portal on the following URL: http://alphapeeler.com/PM/
8. You need to login in above portal with you fast official email id to check the status. Above mentioned URL's password is already communicated to you while registering your FYP project on portal by your supervisor. If you are newly enrolled in FYP, then it is your responsibility to get the authentication details from the supervisor.
9. Online consultation hours:
Students need to communicate to me on Skype after office hours.
Skype ID: abdulmahmood-sss
Time: 6 PM – 09 PM (online consulting hours)
Instructions: Please send your complete roll no, batch, name, section and name of institution while sending the Skype request, otherwise your request may be ignored.
10. Consultation via Email : armahmood786@yahoo.com

Reference books

- a) Design Patterns, Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, 2012.
  b) Refactoring: Improving the Design of Existing Code by Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William Opdyke, don Roberts
  c) Applying UML and Patterns 3rd Edition by Craig Larman
  d) Java Design Patterns by Rohit Joshi, Copyright (c) Exelixis Media P.C., 2015

Assessment

- Final Exam (50%)
- Home work assignments (10%)
- Term exams 30%
- Project(confirmed) 10%

Course Goals

By the end of this course, students will:

- Have a deeper knowledge of the principles of object-oriented design
- Understand the design patterns that are common in software applications
- Understand how these patterns related to object-oriented design
- Use refactoring to facilitate adding new functionality to system
- Use refactoring to improve design
- Refactor existing applications to make them more maintainable
- Recognize when and when not to refactor
- Identify and choose the appropriate type of refactoring technique to solve specific problems

An Introduction to Design Patterns

What is Design Pattern

- Design pattern is a general **reusable** solution to a commonly occurring problem in software design.
- A design pattern is not a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Why Design Patterns

- •To **design a new** software system quickly and efficiently.
- •To **understand a existing** software system.

Introduction

- Promote reuse.
- Use the experiences of software developers.
- A shared library/lingo used by developers.
- “Design patterns help a designer get a design right faster”.

Introduction

- **Based on** the principles of object-oriented programming: abstraction, inheritance, polymorphism and association.
- Are **solutions to recurring problems** to software design.
- Are **independent of the application domain**.
- Example – Variability of interfaces – the model view controller (MVC) pattern.

The Downside

- Although design patterns are useful in promoting flexibility, this maybe at the expense of a more **complicated design**.
- There does not exist
  - A **standardization** for indexing patterns
  - **General practices/processes** for using design patterns during the design process have not as yet been established.

Object-Oriented Principles

- Involves identifying:
  - **Classes** and **objects**
  - What to **encapsulate**
  - **Association** hierarchies
  - **Inheritance** hierarchies
  - **Interface** hierarchies
- Object-oriented designs are evaluated in terms of how <u>**reusable**</u>, <u>**extensible**</u> and <u>**maintainable**</u> they are.

Principles of Object-Oriented Design

- Encapsulate what varies.
- Design/program to an interface and not to an implementation.
- Favour the use composition (association) over inheritance as much as possible.

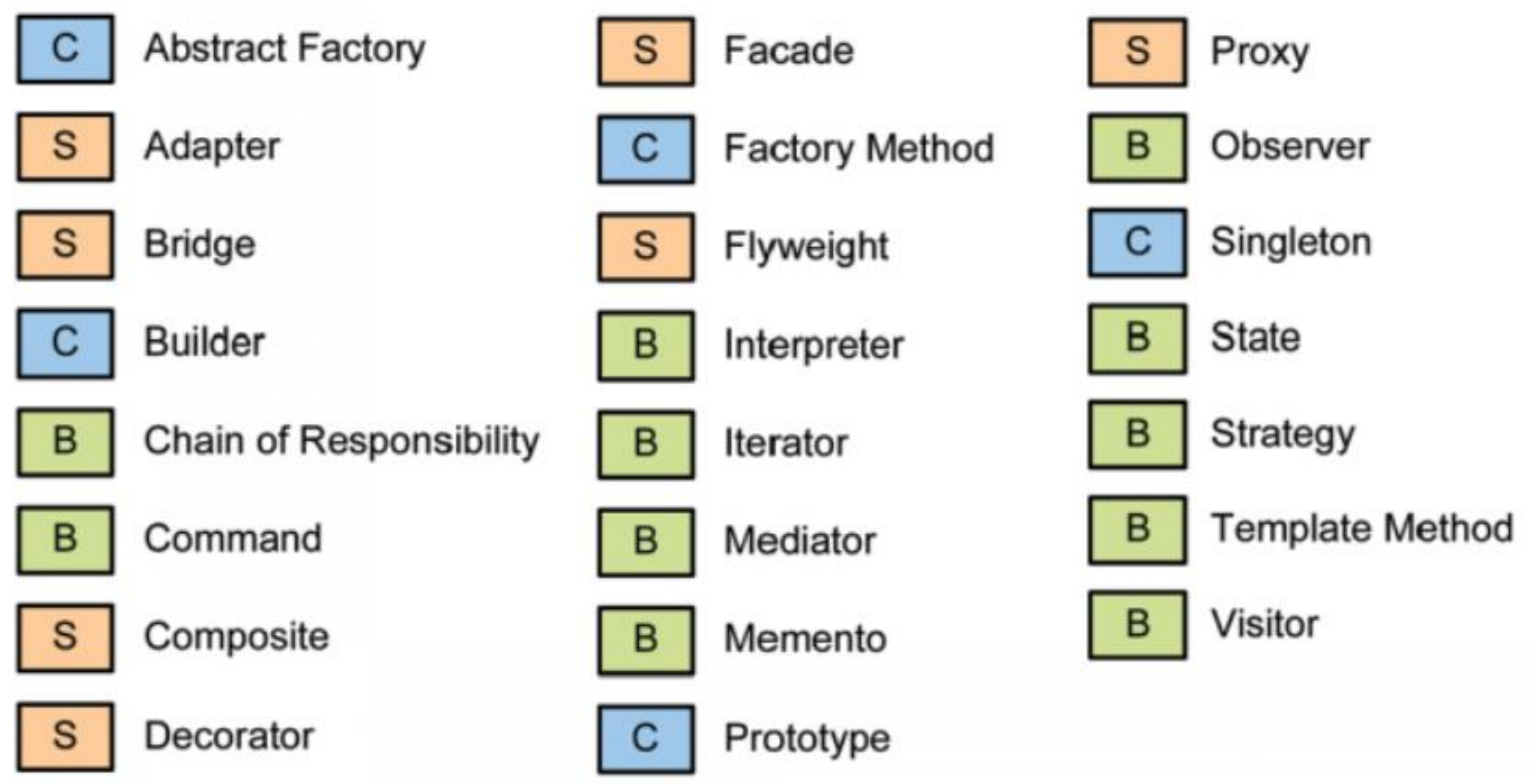
Types of Pattern – Catalog 1

- Creational patterns
  - Focus on <u>Object creation</u>.
  - Focus on the best way to create instances of objects to promote flexibility, e.g. <u>factory pattern.</u>
- Structural patterns
  - Focus on <u>Relationship between entities.</u>
  - Focus on the composition of classes and objects into larger structures, e.g. the <u>adapter pattern.</u>
- Behavioural patterns
  - Focus on <u>Communication between objects</u>
  - Focus on the interaction between classes or objects, e.g. the <u>observer pattern.</u>

Types of Pattern – Catalog 2

- Architectural patterns
  - Focus on the form of the **overall system**.
- Design patterns
  - Focus on the form of the **subsystems** making up the overall system and essentially **provides schemes** for refining them.

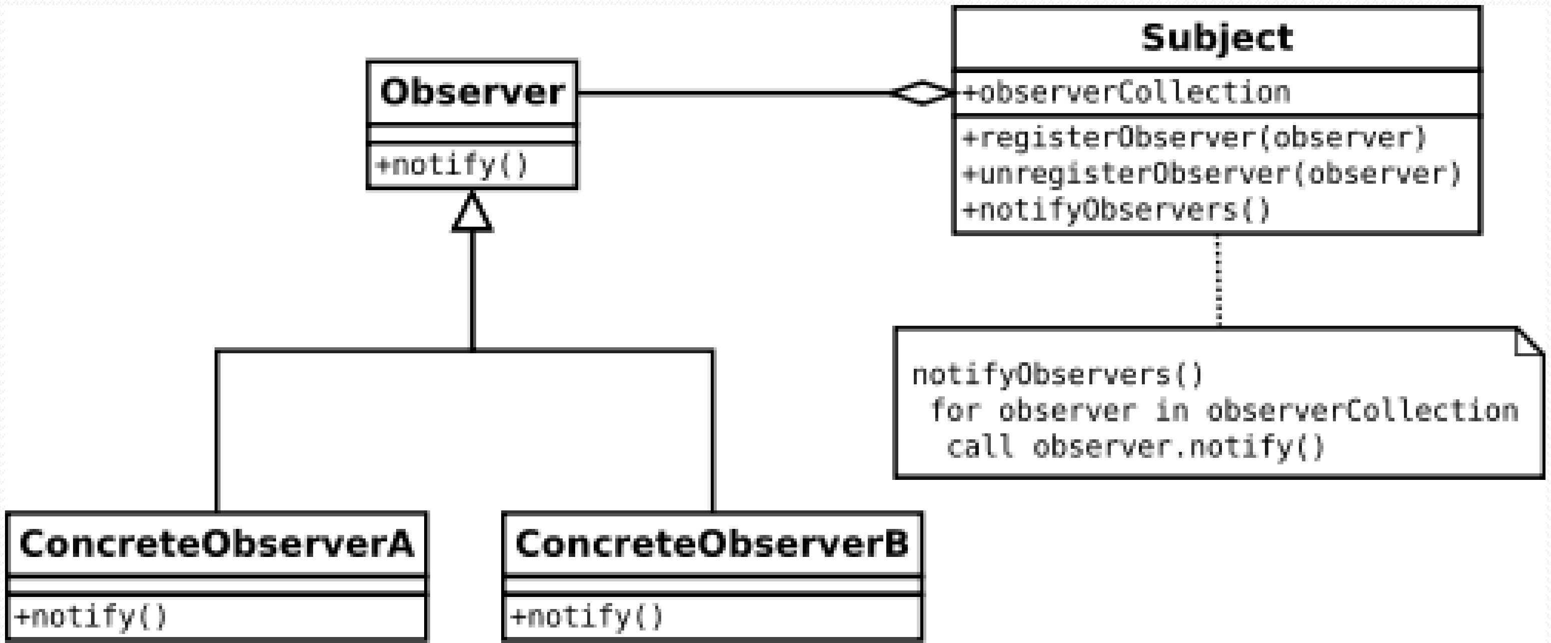
23 GOF DPs



C	Abstract Factory	S	Facade	S	Proxy
S	Adapter	C	Factory Method	B	Observer
S	Bridge	S	Flyweight	C	Singleton
C	Builder	B	Interpreter	B	State
B	Chain of Responsibility	B	Iterator	B	Strategy
B	Command	B	Mediator	B	Template Method
S	Composite	B	Memento	B	Visitor
S	Decorator	C	Prototype		

Observer Design Pattern

- •Observer Design Pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
- •Type : Behavioral pattern.

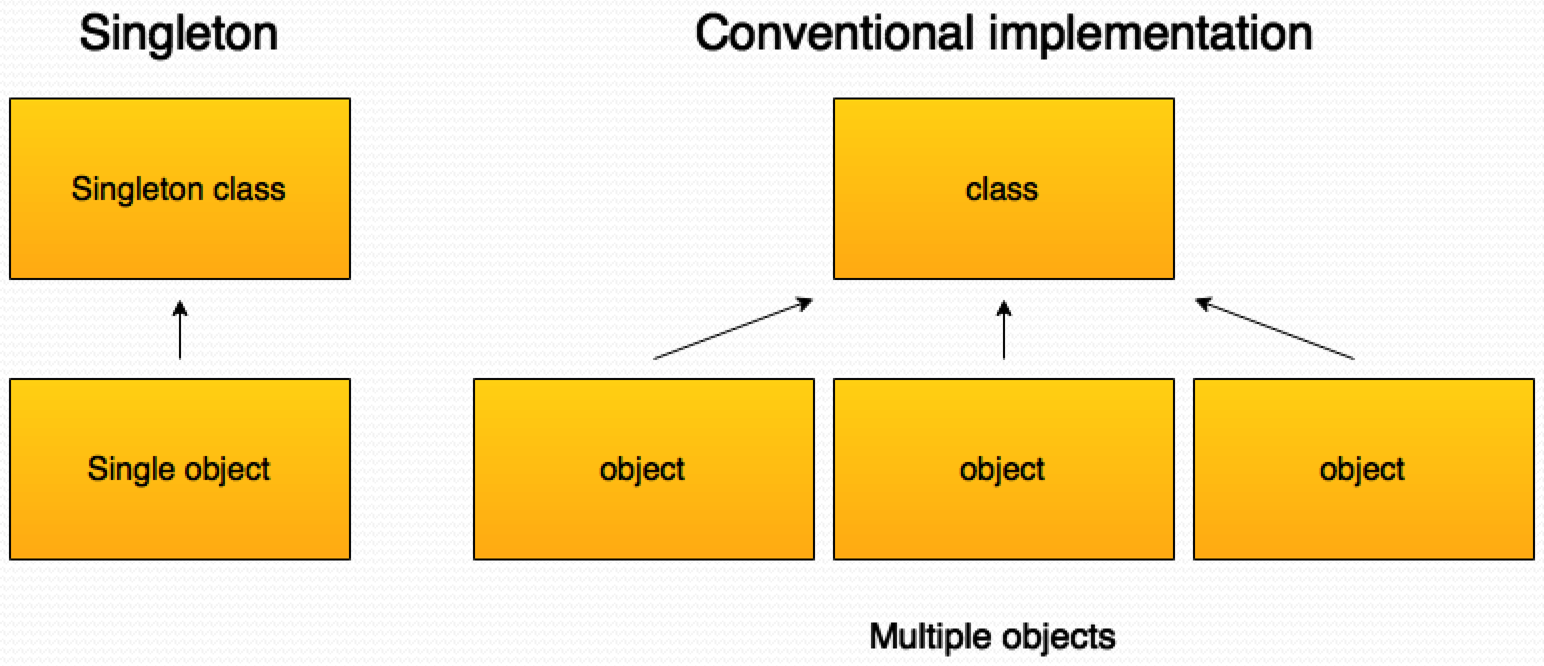


Factory Design Pattern

- Define an interface for creating an object, <u>but let the subclasses decide which class to instantiate.</u> The Factory method lets a class defer instantiation to subclasses.
- Type : Creational pattern.

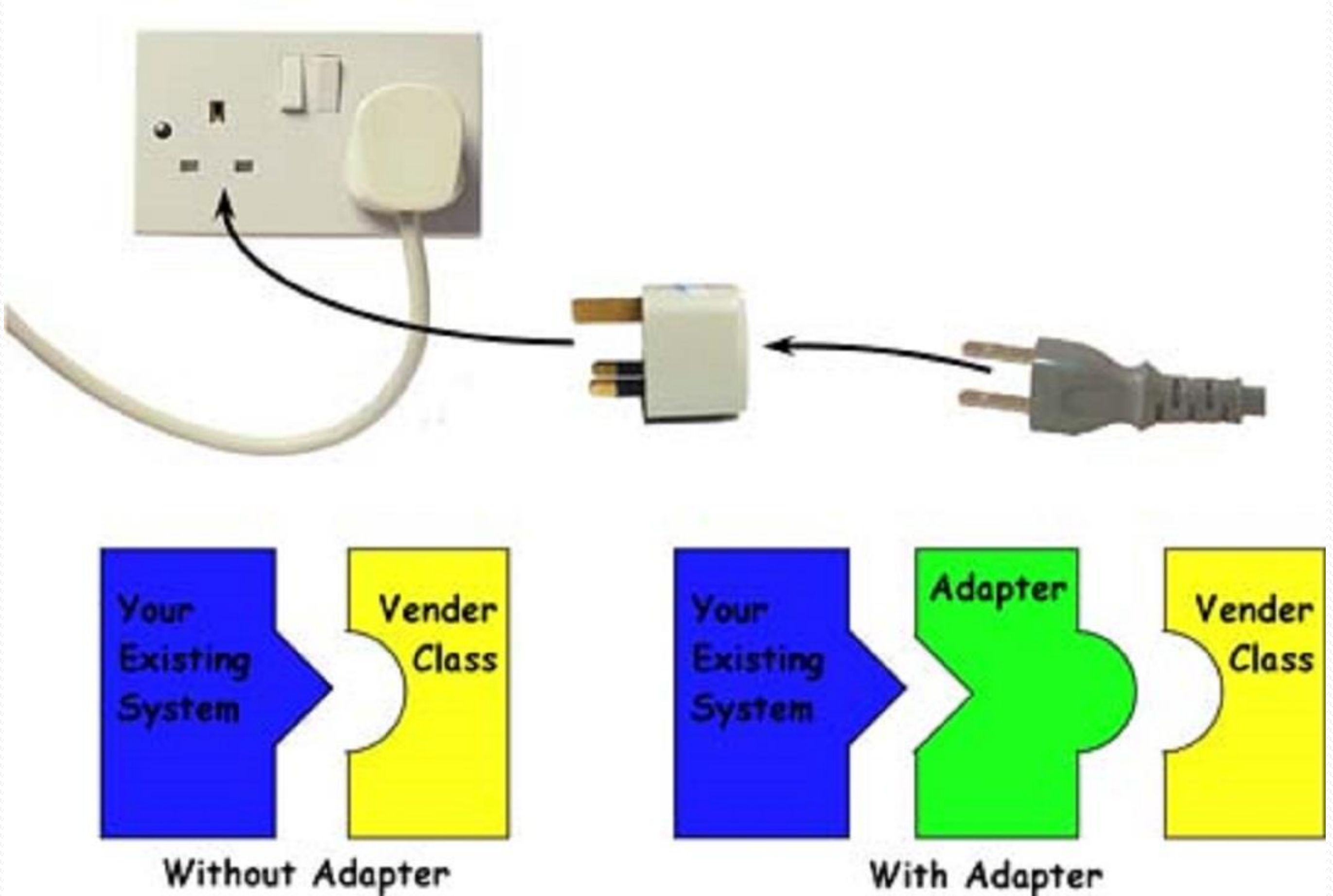
Singleton Design Pattern

- Ensure a class has only one instance, and provide a global point of access to it.
- Encapsulated "just-in-time initialization" or "initialization on first use".
- Type : Creational pattern.



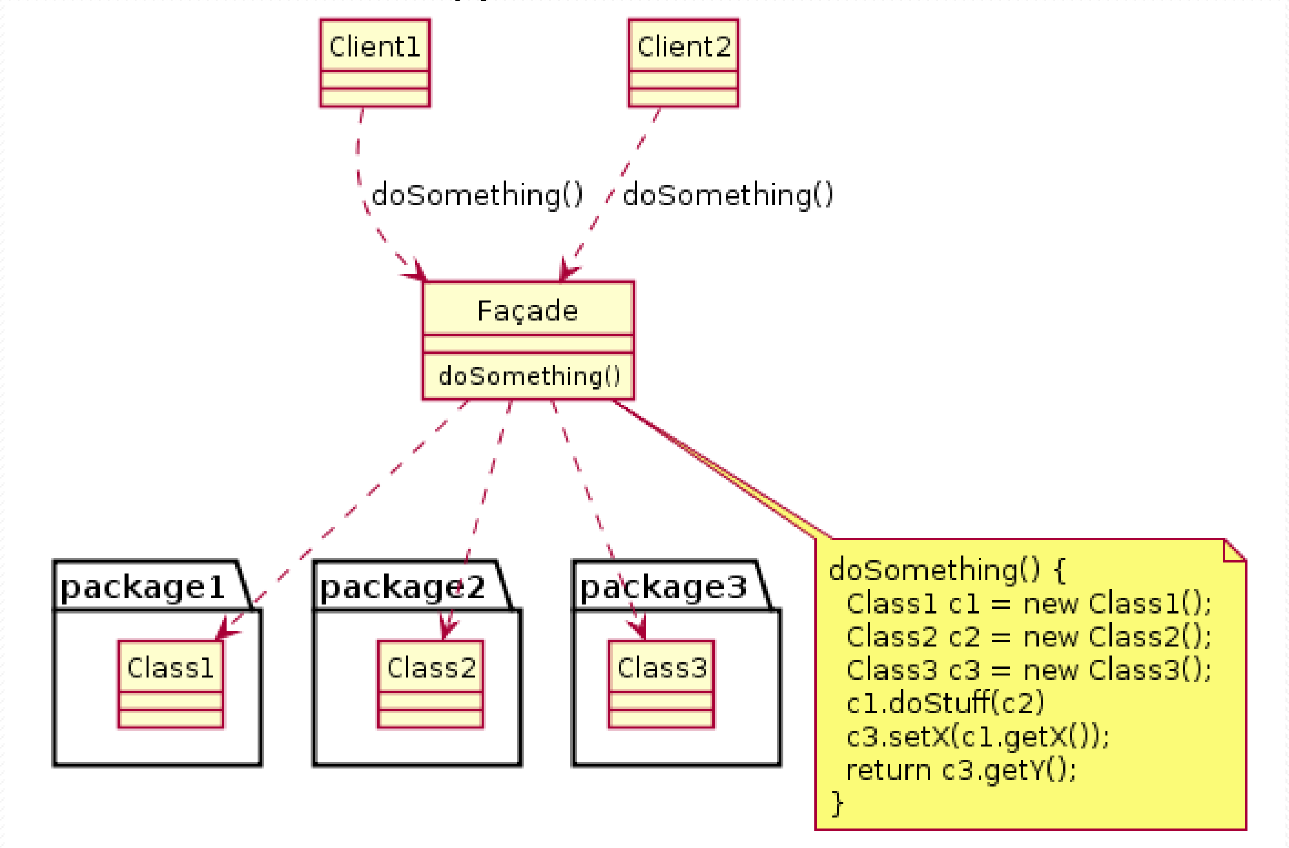
Adaptor Design pattern

- The adapter pattern (often referred to as the wrapper pattern or simply a wrapper) is a design pattern that ***translates*** one interface for a class into a compatible interface.



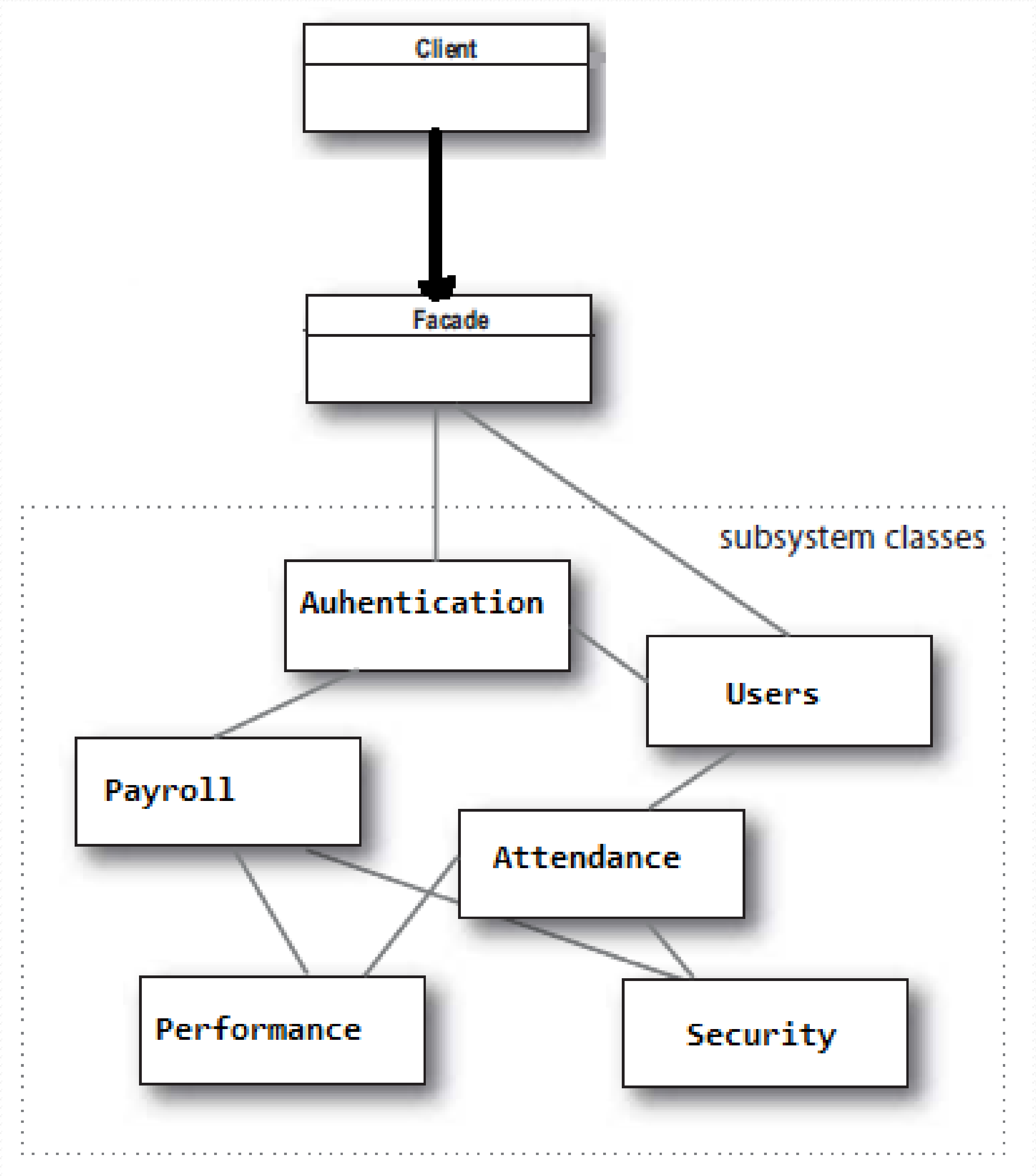
Façade pattern

- A facade is an object that provides a **simplified interface** to a larger body of code, such as a class library.
- Type: Structural Design Pattern.



```
doSomething() {
  Class1 c1 = new Class1();
  Class2 c2 = new Class2();
  Class3 c3 = new Class3();
  c1.doStuff(c2)
  c3.setX(c1.getX());
  return c3.getY();
}
```

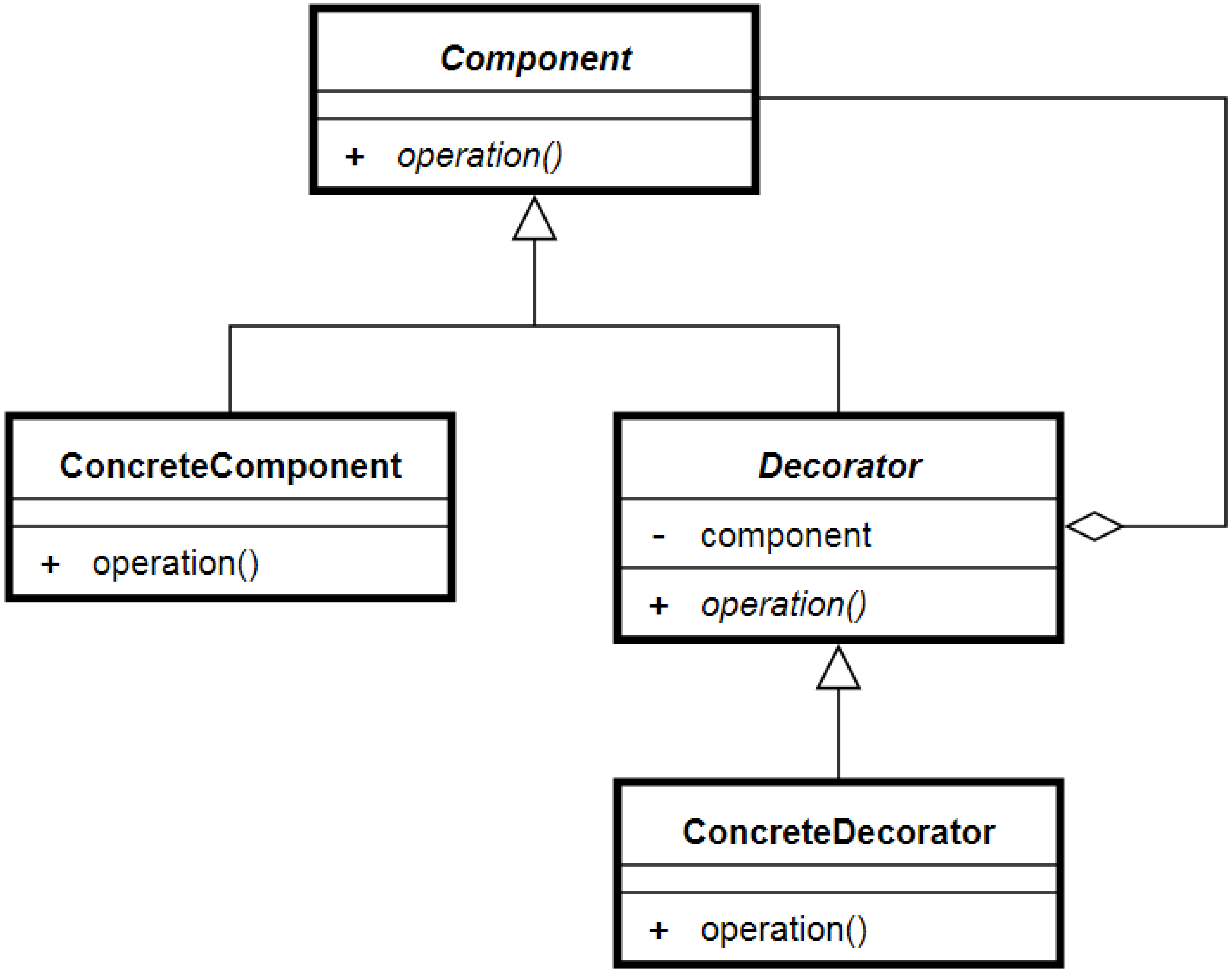
Façade pattern



Relationships between Patterns

- Some patterns are used together, for example the **composite pattern** is sometimes used with iterator or visitor.
- Some patterns are also defined as **alternatives** for others, e.g. the prototype pattern can be used as alternative to the abstract factory pattern.
- Some patterns have **similar designs** , e.g. the composite and decorator pattern.
- **Decorator pattern** allows behavior to be added to an individual object, either statically or dynamically, without affecting the behavior of other objects from the same class.

Decorator pattern



Pattern Scope

- The scope of a pattern specifies whether the pattern applies to **classes** or **objects**.
- **Class patterns** describe relationships between classes and their subclasses. These relationships are static.
- **Object patterns** describe the relationships between objects. Theses relationships can be changed at runtime.

Defining Patterns

- Are defined in terms of classes and objects and relationships between them.
- Using existing well-tested patterns saves time instead of deriving them from scratch each time.
- Patterns may consist of smaller patterns/sub-patterns.
- Class diagrams are used to express design patterns.

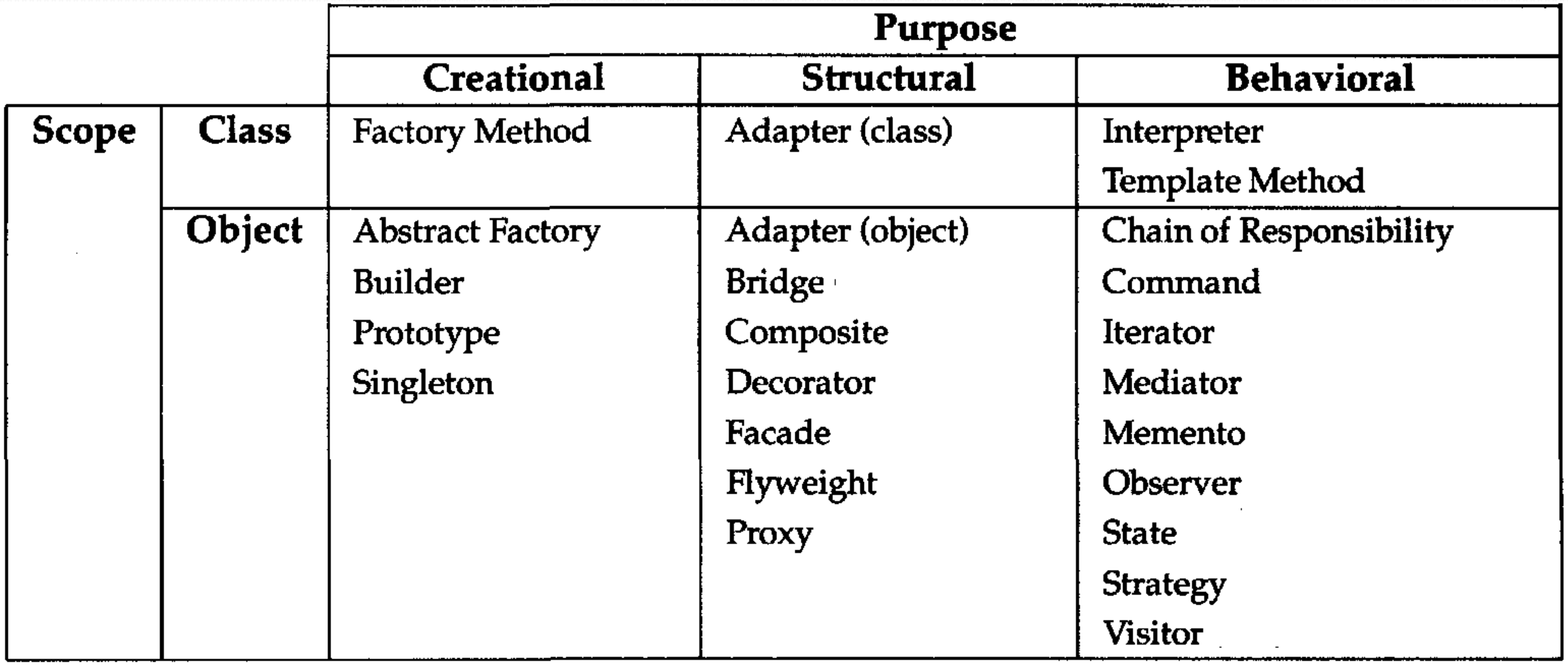
Core Components of a Pattern

- The problem which the pattern was used to solve and the situation giving rise to the problem. A **list of the conditions that must be met** in order to apply the pattern may also be included.
- The **core solution** to the problem in terms of a description of the design rather than implementation details.
- **Uses** of the solution

A More Detailed Definition

- Name
- Intent of the pattern
- Aliases
- The problem
- Solution
- Example/s
- Applicability
- Structure
- Participants
- Collaborations
- Implementation
- Sample code
- Known uses
- Related patterns
- Consequences

Design pattern space



		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method	Adapter (class)	Interpreter Template Method
	Object	Abstract Factory Builder Prototype Singleton	Adapter (object) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

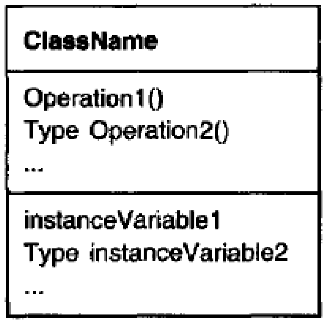
How Design Patterns Solve Design Problems

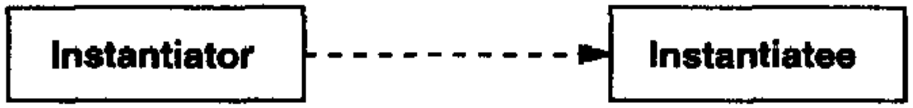
- (1) <u>Finding</u> appropriate objects.
- (2) Determining the <u>granularity</u> of objects.
- (3) Specifying object <u>interfaces</u> – definition of interfaces and relationships between them.

- **Signature:** Object operation name, parameters, and its return value.
- **Interface:** The set of all signatures defined by an object's operations

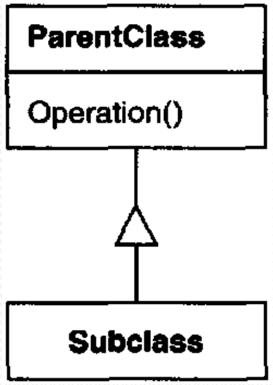
How Design Patterns Solve Design Problems

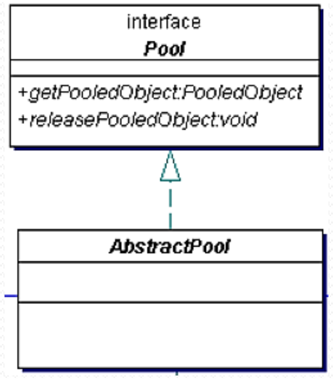
- (4) Specifying object implementations.

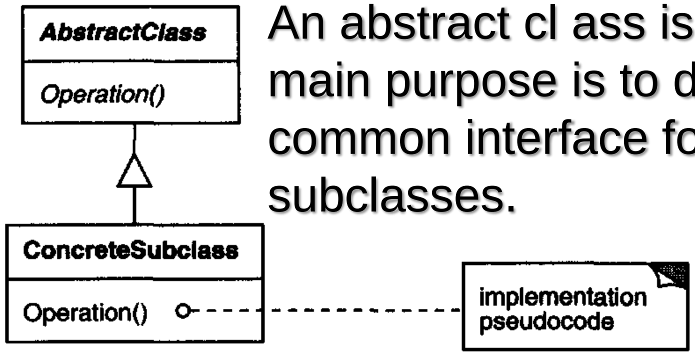




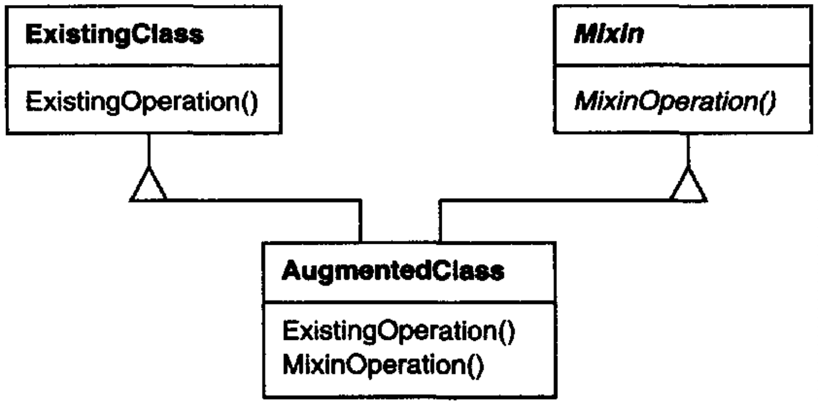
A dashed arrowhead line indicates a class that instantiates objects o f another class. The arrow points to the class of the instantiated objects.







An abstract cl ass is one whose main purpose is to define a common interface for its subclasses.



A mixin (or mix-in) is a class that contains methods for use by other classes without having to be the **parent** class of those other classes.

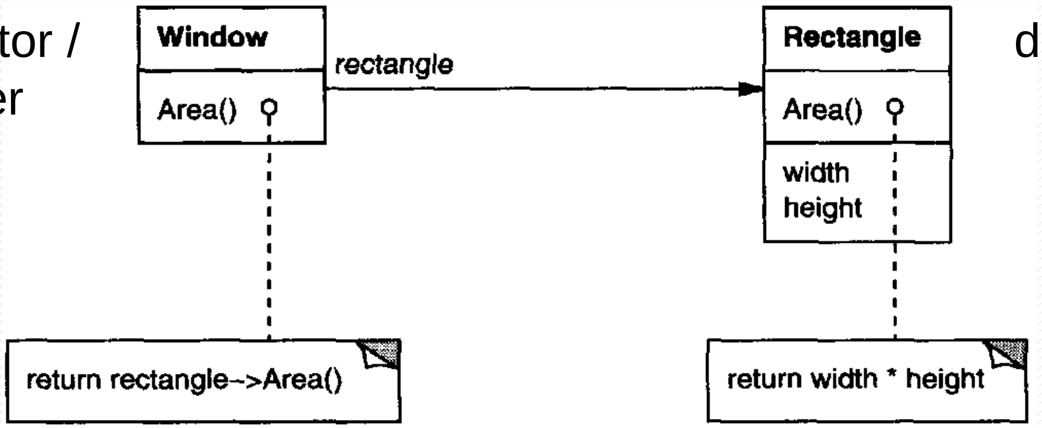
How Design Patterns Solve Design Problems

- **Inheritance vs Composition**
  - white-box reuse vs black-box reuse
    - Black-box reuse: Object composition requires that the objects being composed have well-defined interfaces. No internal details of objects are visible. Objects appear only as "black boxes.“
- **Interface vs Abstract classes**
  - Abstract classes are used when we require classes to share a similar behavior (or methods). However, if we need classes to share method signatures, and not the methods themselves, we should use interfaces.
  - A class can only inherit from one abstract class at a time
  - Interfaces are used to implement the concept of multiple inheritance in object
  - Because an interface is not a class, it does not allow access modifiers
  - An interface is just an empty signature and does not contain a body (code)

How Design Patterns Solve Design Problems

- (5) Using <u>reuse mechanisms</u> – delegation and parameterised types.
  - Delegation is a way of making composition as powerful for reuse as inheritance
    - receiving object delegates operations to its delegate.

delegator / receiver



delegate

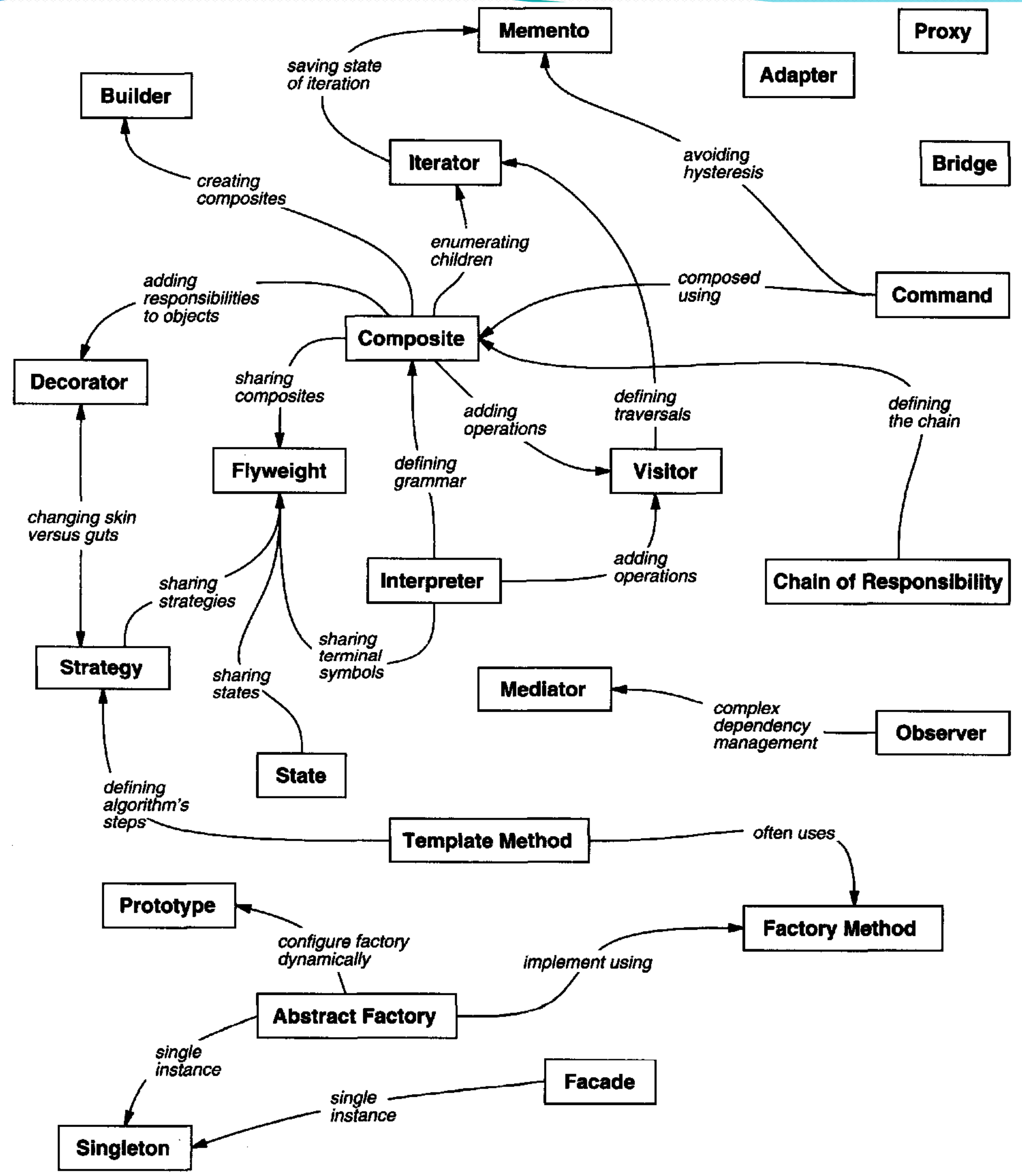
  - Inheritance versus Parameterized Types
    - Another technique for reusing functionality is through parameterized types, also known as generics (Ada, Eiffel) and templates (C++).
- (6) Relating <u>runtime</u> and <u>compile-time</u> structures

Inheritance vs Delegation

```
class RealPrinter { // Base class
    void print() {
        System.out.println("Printing Data");
    }
}
class Printer extends RealPrinter { // Child
    void print() {
        super.print();
    }
}
public class Tester {
    public static void main(String[] args) {
        Printer printer = new Printer();
        printer.print();
    }
}
```

```
class RealPrinter {  // the "delegate"
    void print() {
        System.out.println("The Delegate");
    }
}
class Printer { // delegator or receiver
    RealPrinter p = new RealPrinter();
    // create the delegate
    void print() {
        p.print(); // delegation
    }
}
public class Tester {
    public static void main(String[] args) {
        Printer printer = new Printer();
        printer.print();
    }
}
```

Design pattern relationships



Designing for Change

- A design must <u>facilitate reuse </u>and change.
- We need to design so as to <u>avoid redesign</u>.
- Design patterns can be used to prevent particular causes of redesign.

Avoiding Redesign

- Explicitly declaring class instances instead of using an interface -abstract factory, factory method and prototype patterns.
- Dependence of specific operations, i.e. using hard-coded requests - chain of responsibility and the command patterns.
- Limit software and hardware platform dependencies - abstract factory and bridge patterns.
- Change in Object representations / implementations - abstract factory, bridge, memento and proxy.

Avoiding Redesign

- Algorithmic dependencies - Algorithms that are likely to change should be isolated from the definition of the objects using them -builder, iterator, strategy, template and visitor patterns.
- Avoid Tight coupling - Tight coupling does not facilitate reuse- abstract factory, bridge, chain of responsibility, command, facade, mediator and observer patterns.
- Subclassing to extend functionality - Rather than using inheritance or association it maybe more efficient to combine both by creating one subclass that is associated with existing class - bridge, chain of responsibility, composite, decorator, observer and strategy classes.
- Difficulty in altering classes - In some cases adapting a class maybe difficult, e.g. the source code may not be available adapter, creator and visitor patterns.

Applying a Pattern

- In designing a system different patterns are used to design the different aspects of the system.
- Design patterns allow parts of the system to vary independently of other parts of the system.
- Patterns are often combined. Using one pattern my introduce further patterns into the design.
- Methods for applying design patterns and deciding which one to use.

Breaking Down the Problem

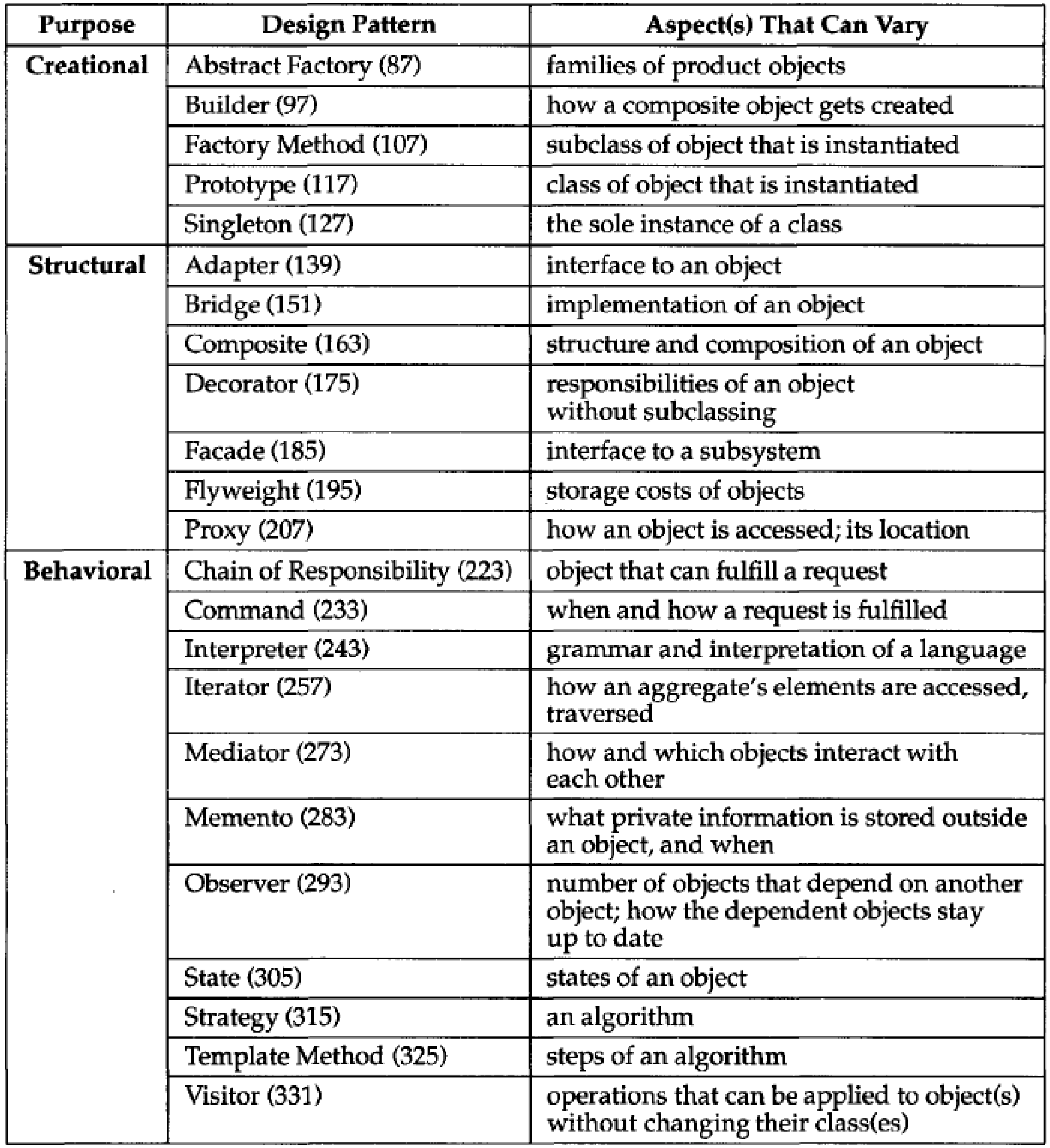
- Describe the problem and its subproblems.
- Select the category of patterns that is suitable for the design task.
- Compare the problem description with each pattern in the category.
- Identify the benefits and disadvantages of using each of the patterns in the category.
- Choose the pattern that best suits the problem.

Choosing a Design Pattern

- Consider the problems solved by design problems and what solution is needed for the problem at hand.
- Consider the intent of each pattern and which is most similar to the problem at hand.
- Analyse the relationships between patterns to determine which is the correct group of patterns to use.
- Determine whether creational, structural or behavioural patterns are needed and which of the patterns in the most suitable category is/are relevant to the problem at hand.
- Look at what could cause redesign for the problem at hand and patterns that can be used to avoid this.
- Identify which aspect/s of the system need to varied independently and which patterns will cater for this.

Using a Design Pattern

- Obtain an overview of the pattern.
- Obtain an understanding of the classes and objects and relationships between them.
- Choose application-specific names for the components of the patterns.
- Define the classes.
- Choose application-specific names for the operations defined in the pattern.
- Implement the necessary operations and relationships.



Purpose	Design Pattern	Aspect(s) That Can Vary
Creational	Abstract Factory (87)	families of product objects
	Builder (97)	how a composite object gets created
	Factory Method (107)	subclass of object that is instantiated
	Prototype (117)	class of object that is instantiated
	Singleton (127)	the sole instance of a class
Structural	Adapter (139)	interface to an object
	Bridge (151)	implementation of an object
	Composite (163)	structure and composition of an object
	Decorator (175)	responsibilities of an object without subclassing
	Facade (185)	interface to a subsystem
	Flyweight (195)	storage costs of objects
	Proxy (207)	how an object is accessed; its location
Behavioral	Chain of Responsibility (223)	object that can fulfill a request
	Command (233)	when and how a request is fulfilled
	Interpreter (243)	grammar and interpretation of a language
	Iterator (257)	how an aggregate's elements are accessed, traversed
	Mediator (273)	how and which objects interact with each other
	Memento (283)	what private information is stored outside an object, and when
	Observer (293)	number of objects that depend on another object; how the dependent objects stay up to date
	State (305)	states of an object
	Strategy (315)	an algorithm
	Template Method (325)	steps of an algorithm
	Visitor (331)	operations that can be applied to object(s) without changing their class(es)

Inheritance versus composition

- When to use Inheritance
  - A person *is a* human.
  - A cat *is an* animal.
  - A car *is a* vehicle.

- When to use composition
  - A car *has a* battery (a battery *is part of* a car).
  - A person *has a* heart (a heart *is part of* a person).
  - A house *has a* living room (a living room *is part of* a house).

Inheritance versus composition

Inheritance

```
class Vehicle {
    String brand;
    String color;
    double weight;
    double speed;
    void move() {
        System.out.println("The vehicle is moving");
    }
}
public class Car extends Vehicle {
    String licensePlateNumber;
    String owner;
    String bodyStyle;
    public static void main(String[] args) {
        System.out.println(new Vehicle().brand);
        System.out.println(new Car().brand);
        new Car().move();
    }
}
```

Inheritance versus composition

composition

```
public class CompositionExample {
    public class Bedroom { }
    public class LivingRoom { }
    public class House {
        Bedroom bedroom;
        LivingRoom livingRoom;
        House(Bedroom bedroom, LivingRoom livingRoom) {
            this.bedroom = bedroom;
            this.livingRoom = livingRoom;
        }
    }
    public static void main(String[] args) {
        new House(new Bedroom(), new LivingRoom());
        // House is composed of Bedroom & LivingRoom
    }
}
```

Method overriding with Java inheritance

```
class Animal {
    void emitSound() {
        System.out.println("The animal sound");
    }
}
class Cat extends Animal {
    @Override
    void emitSound() {
        System.out.println("Meow");
    }
}
class Dog extends Animal {
}
public class Main {
    public static void main(String[] args) {
        Animal cat = new Cat();
        Animal dog = new Dog();
        Animal animal = new Animal();
        cat.emitSound();
        dog.emitSound();
        animal.emitSound();
    }
}
```

```
Output:
Meow
The animal sound
The animal sound
```

Does Java have multiple inheritance?

- If you attempt multiple inheritance like I have below, the code won't compile:

```
class Animal {}
class Mammal {}
class Dog extends Animal, Mammal {}
```

- One solution using classes would be to inherit one-by-one: (Multi-Level)

```
class Animal {}
class Mammal extends Animal {}
class Dog extends Mammal {}
```

- Another solution is to replace the classes with interfaces:

```
interface Animal {}
interface Mammal {}
class Dog implements Animal, Mammal {}
```