# **GRASP**

**GRASP** – describes fundamental principles of OOP design.

* **G**eneral
* **R**esponsibility
* **A**ssignment
* **S**oftware
* **P**atterns

OOP principles -> GRASP -> GoF

GRASP has 9 principles and they are formed from OOP

|  |  |  |  |
| --- | --- | --- | --- |
| ID | PRINCIPLE | DESCRIPTION |  |
| 1 | **CREATOR** | What problem solves – who should be responsible for creating object?  Answer – class that uses that object should create it. |  |
| 2 | **INFORMATION**  **EXPERT** | What problem solves – who should be responsible for creating object?  Answer – class that uses that object should create it.  Information Expert – is a class that holds all info about the class |  |
| 3 | **CONTROLLER** | What problem solves – who is a responsible to handle incoming system events in a system?  Answer – all incoming events must be accumulated in one place This place is called a controller and separates from a business logic. Then Controller delegates tasks to classes | Example, UI sends a lot of request like authentication, inputs and how it should come in a system? It should pass through controller |
| 4 | **LOW COUPLING** | What problem solves – we need to provide as possible low number dependencies between systems?  Answer – we need to share responsibilities in such way that provides low coupling.  Coupling is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements. | low coupling is desirable because a change in one area of an application will require less changes throughout the entire application. |
| 5 | **HIGH COHESION** | What problem solves – provides a single responsibility. Each class must be responsible only for one responsibility in a system  Answer – we need to share responsibilities in such way that provides low coupling.  Cohesion is a measure how strongly all responsibilities of the element are related. In other words, what is the degree to which the parts inside a element belong together. |  |
| 6 | **POLYMORPHISM** | Polymorphism is used to create pluggable components  What problem solves  1) there are few ways how system can behave  2) how switch on system that consists of module components  Answer – We can use polymorphism instead if if/else expressions |  |
| 7 | **PURE FABRICATIONS** | What problem solves: if you want make sure your classes have low coupling and high cohesion you should create an additional class  Answer – it creates an additional class to provide low coupling and high cohesion. |  |
| 8 | **INDIRECTION** | What problem solves: how to minimize dependencies of two objects. For example, one class uses methods of another one  Answer – any object in the code must be called through its interface. |  |
| 9 | **PPROTECTED VARIATIONS** | What problem solves – how correctly design a system in order to changes of one component do not impact another components  Answer – find a place of possible changes and share responsibilities to provide stable work of system where . |  |

# **#1. CREATOR - PRINCIPLE**

[Creator] identifies what class should be responsible for creating a new instance of a class

**What problem solves – who should be responsible for creating object?**

**Answer – class that uses that object should create it**.

Pros: not increase coupling

Cons: can be complicated

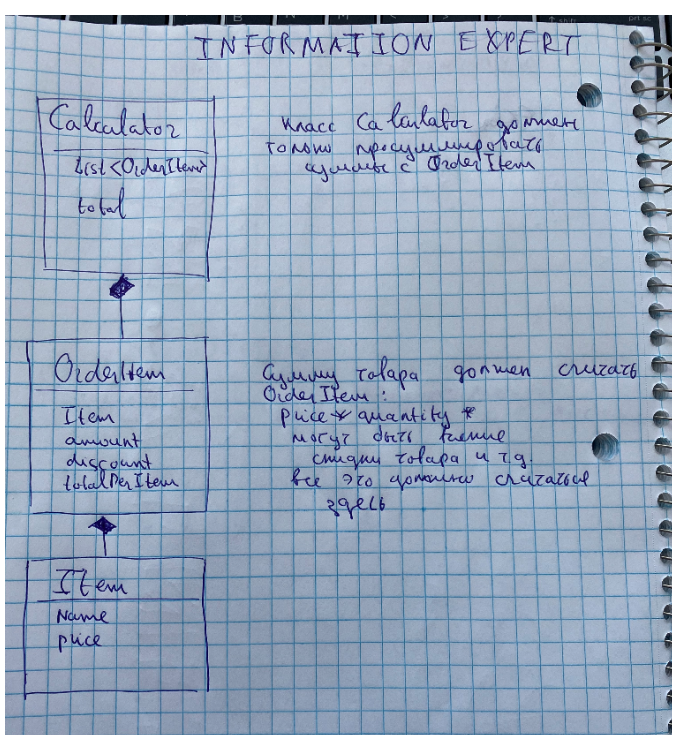
# **#2. INFORMATION EXPERT - PRINCIPLE**

**What problem solves – delegation of responsibilities, for example, calculation of total price**?

**Answer – calculate/process info should that component that contains that info**.

**Information Expert** – is a class that holds all info about the class

For example, you have an example of store system. Who is responsible for creating a total of check? OrderItem



# **#3. CONTROLLER– PRINCIPLE**

**What problem solves – who is a responsible to handle incoming system events in a system**?

**Answer – all incoming events must be accumulated in one place This place is called a controller and separates from a business logic**. Then Controller delegates tasks to classes

Example, UI sends a lot of request like authentication, inputs and how it should come in a system? It should pass through controller

Controller – forms multithreading stream in one stream

# **#4. LOW COUPLING- PRINCIPLE**

**What problem solves – we need to provide as possible low number dependencies between systems**?

**Answer – we need to share responsibilities in such way that provides low coupling**.

**Coupling** is a measure of how strongly one element is connected to, has knowledge of, or relies on other elements.

low coupling is desirable because a change in one area of an application will require less changes throughout the entire application.

# **#5. – HIGH COHESION PRINCIPLE**

**What problem solves – provides a single responsibility. Each class must be responsible only for one responsibility in a system**

**Answer – we need to share responsibilities in such way that provides low coupling**.

[**Cohesion**](https://en.wikipedia.org/wiki/Cohesion_(computer_science))is a measure how strongly all responsibilities of the element are related. In other words, what is the degree to which the parts inside a element belong together.

# **#6. – POLYMORPHISM PRINCIPLE**

Polymorphism is used to create pluggable components

**What problem solves**

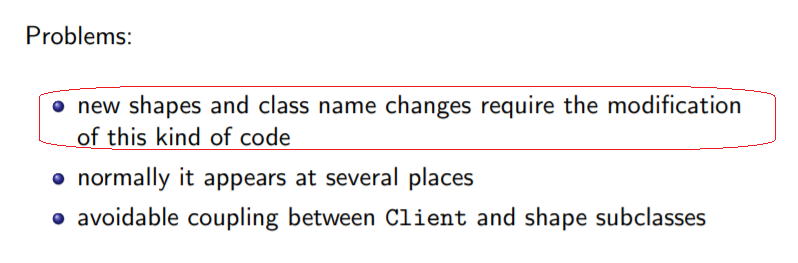
1. **there are few ways how system can behave**
2. how switch on system that consists of module components

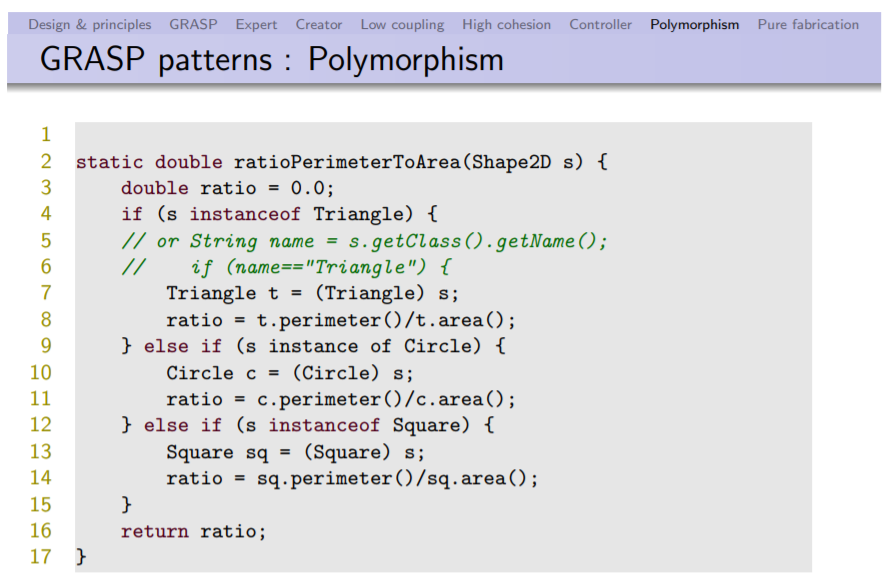
**Answer –** **We can use polymorphism instead if if/else expressions**

Grasp pattern [polymorphism] is an implementation of OOP principle [polymorphism]

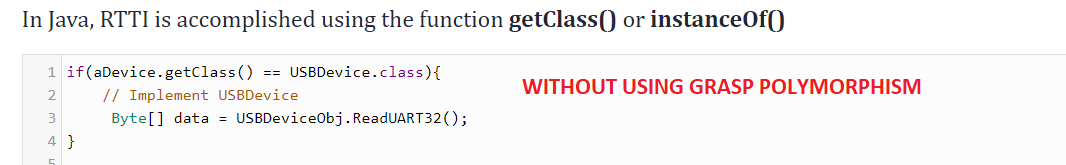
pros: allows more easy extends system by new functional

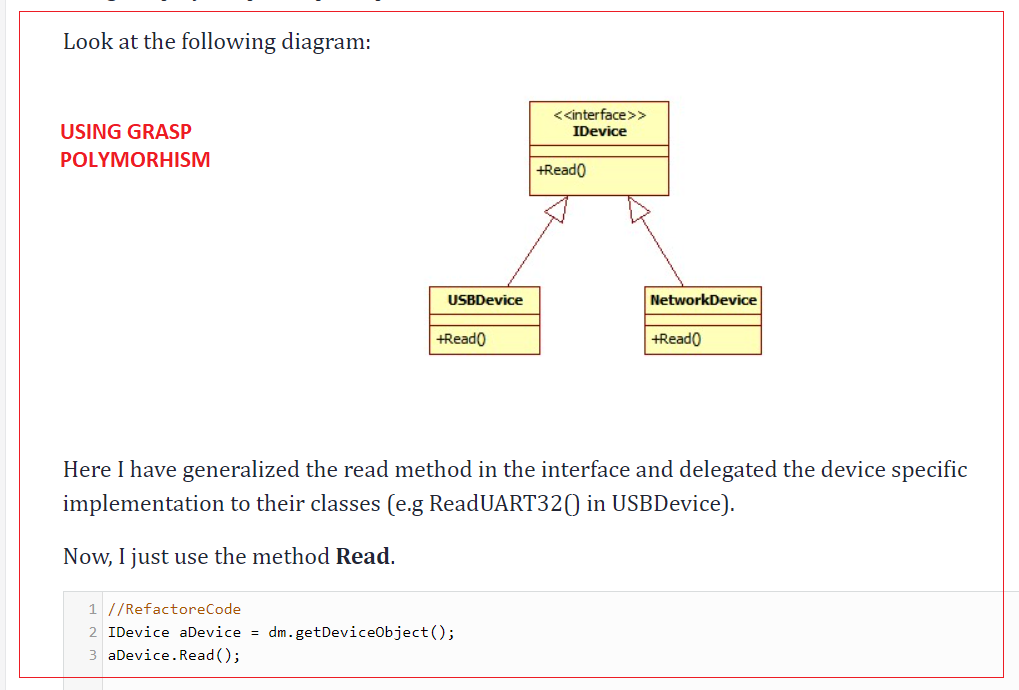
cons: should thinking carefully before applying this principle



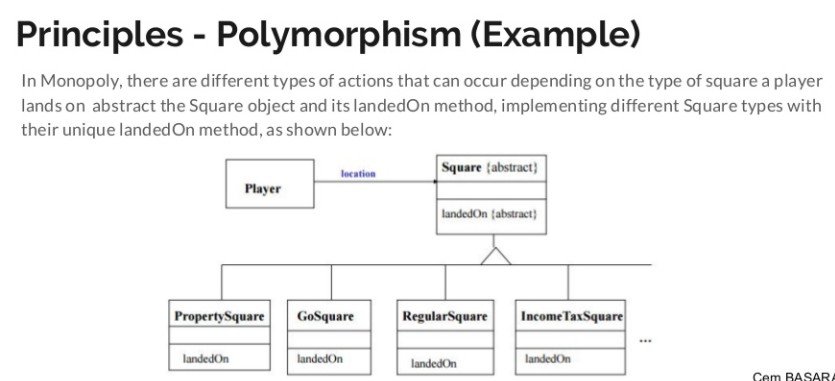


Example1





Example2



# **#7. PURE FABRICIATION PRINCIPLE**

Used very rarely

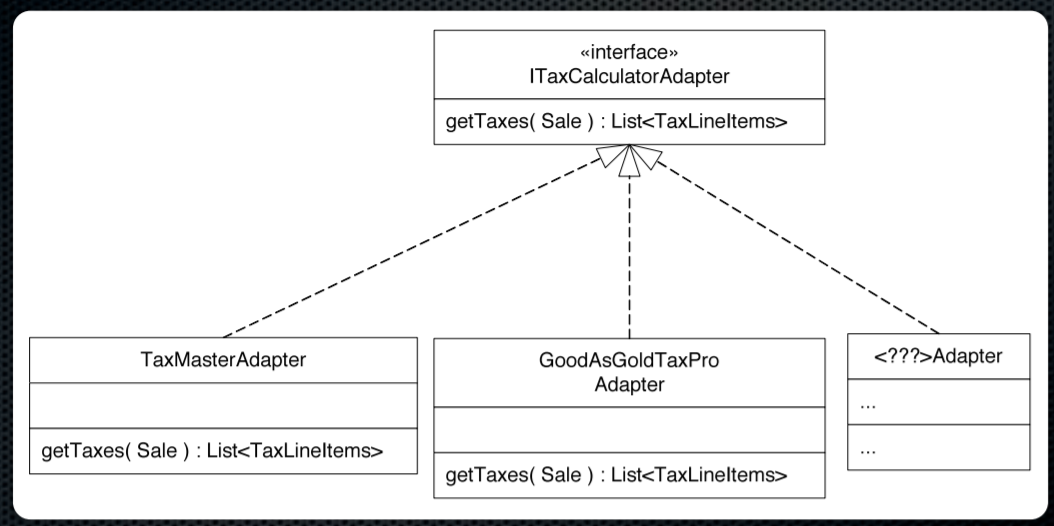
**What problem solves: if you want make sure your classes have low coupling and high cohesion you should create an additional class**

**Answer – it creates an additional class to provide low coupling and high cohesion**.

# **#8. – INDIRECTION PRINCIPLE**

**What problem solves: how to minimize dependencies of two objects. For example, one class uses methods of another one**

**Answer –** any object in the code must be called through its interface.



# **#9 . – PROTECTED VARIATIONS PRINCIPLE**

**What problem solves – how correctly design a system in order to changes of one component do not impact another components**

**Answer – find a place of possible changes and share responsibilities to provide stable work of system where** .

This principles combines all principles above