CSC301: Introduction to Software Engineering

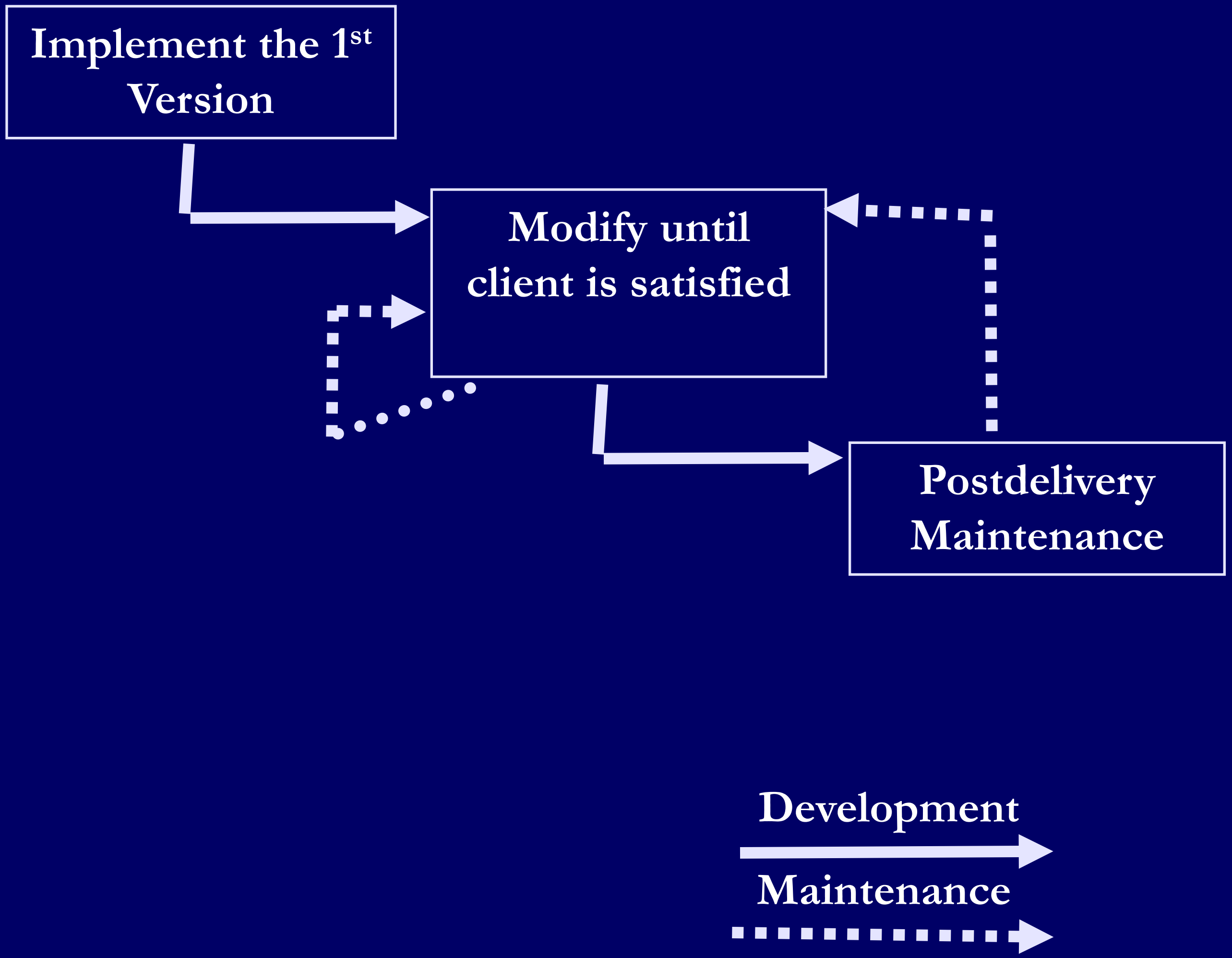
Lecture 5

Wael Aboulsaadat

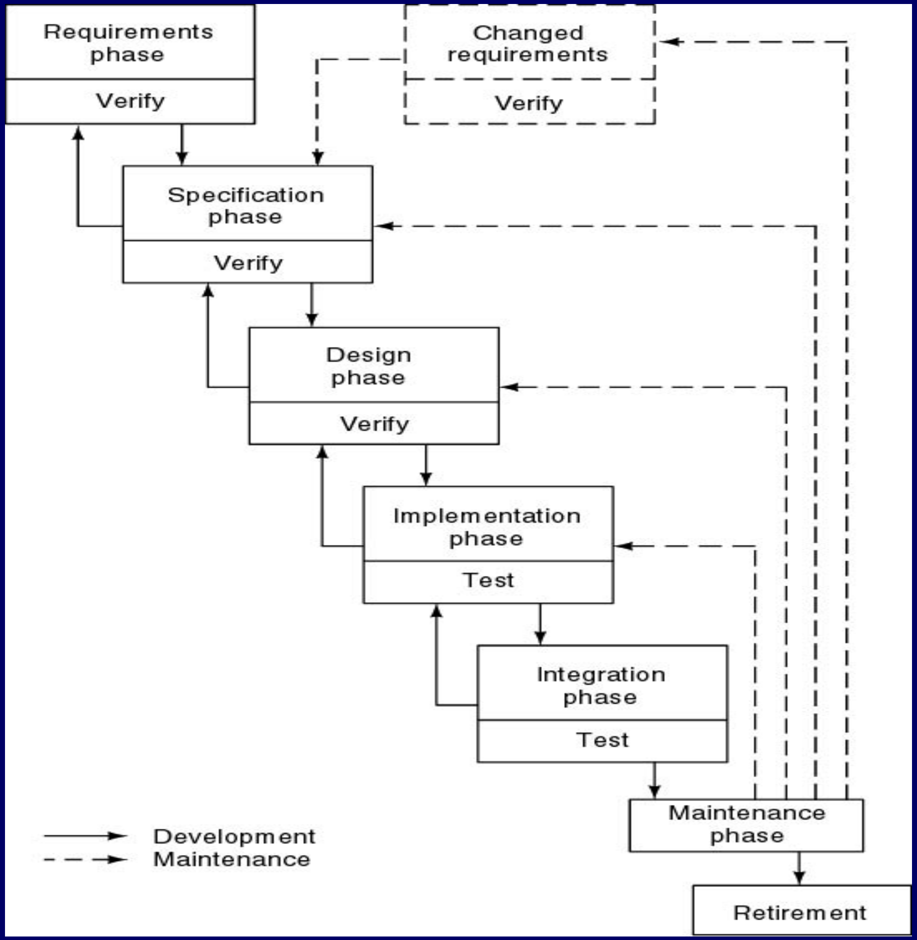
Software Development Lifecycle SDLC

Code-and-Fix Model

- No design
- No specifications
  - Maintenance nightmare
- The easiest way to develop software
- The most expensive way
- Typically used by a start-up…

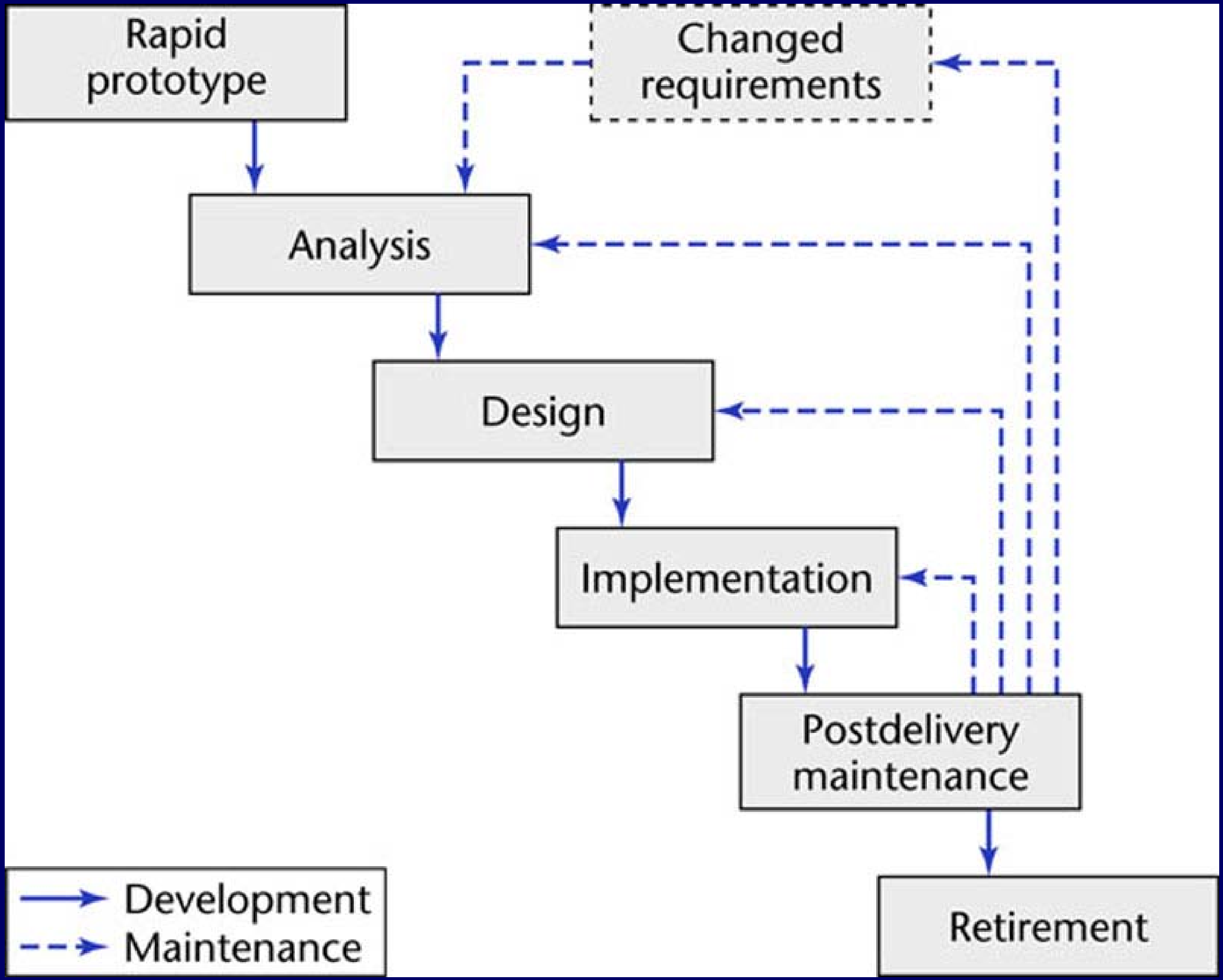


Waterfall model: Linear & Sequential

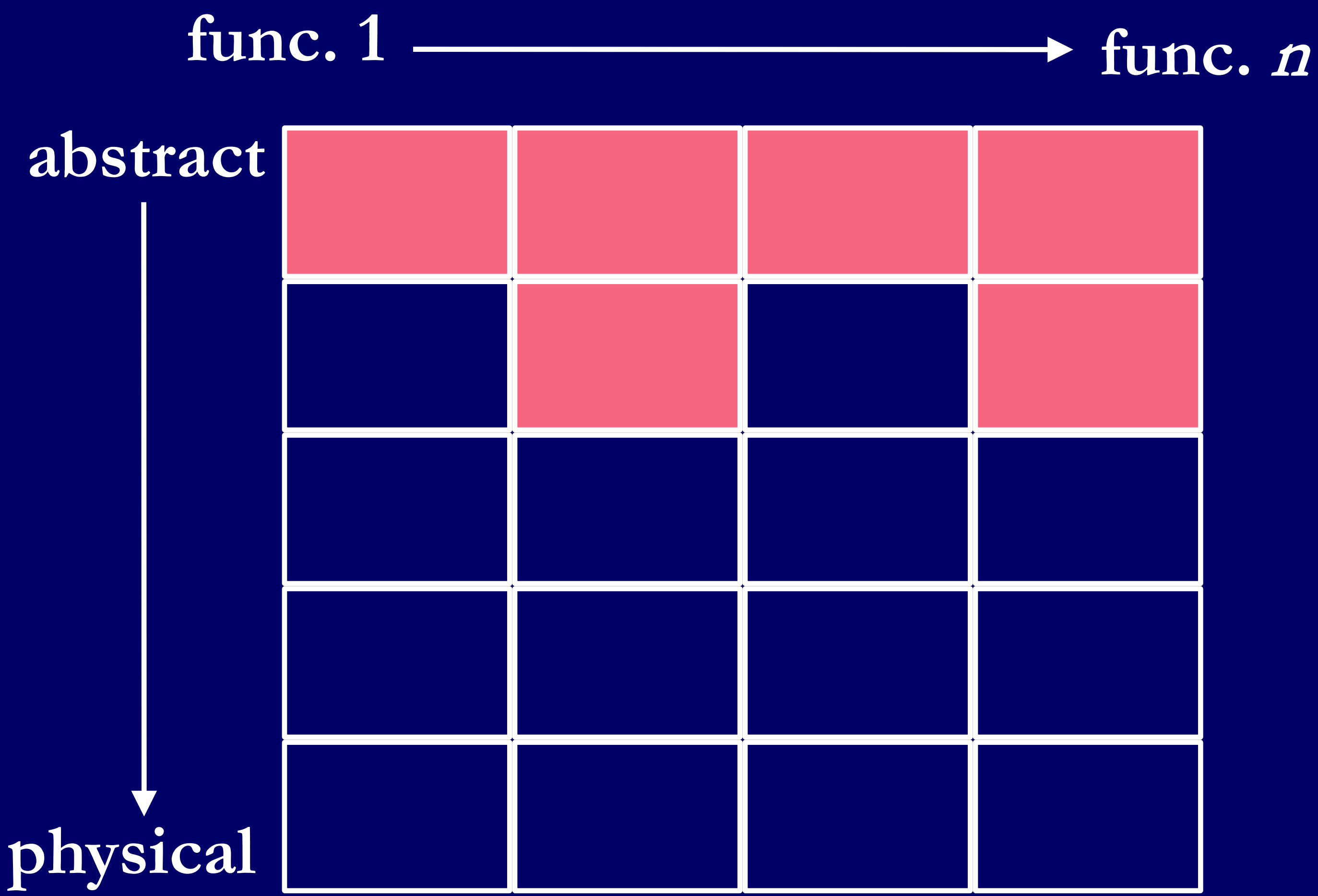


Rapid Prototyping Model

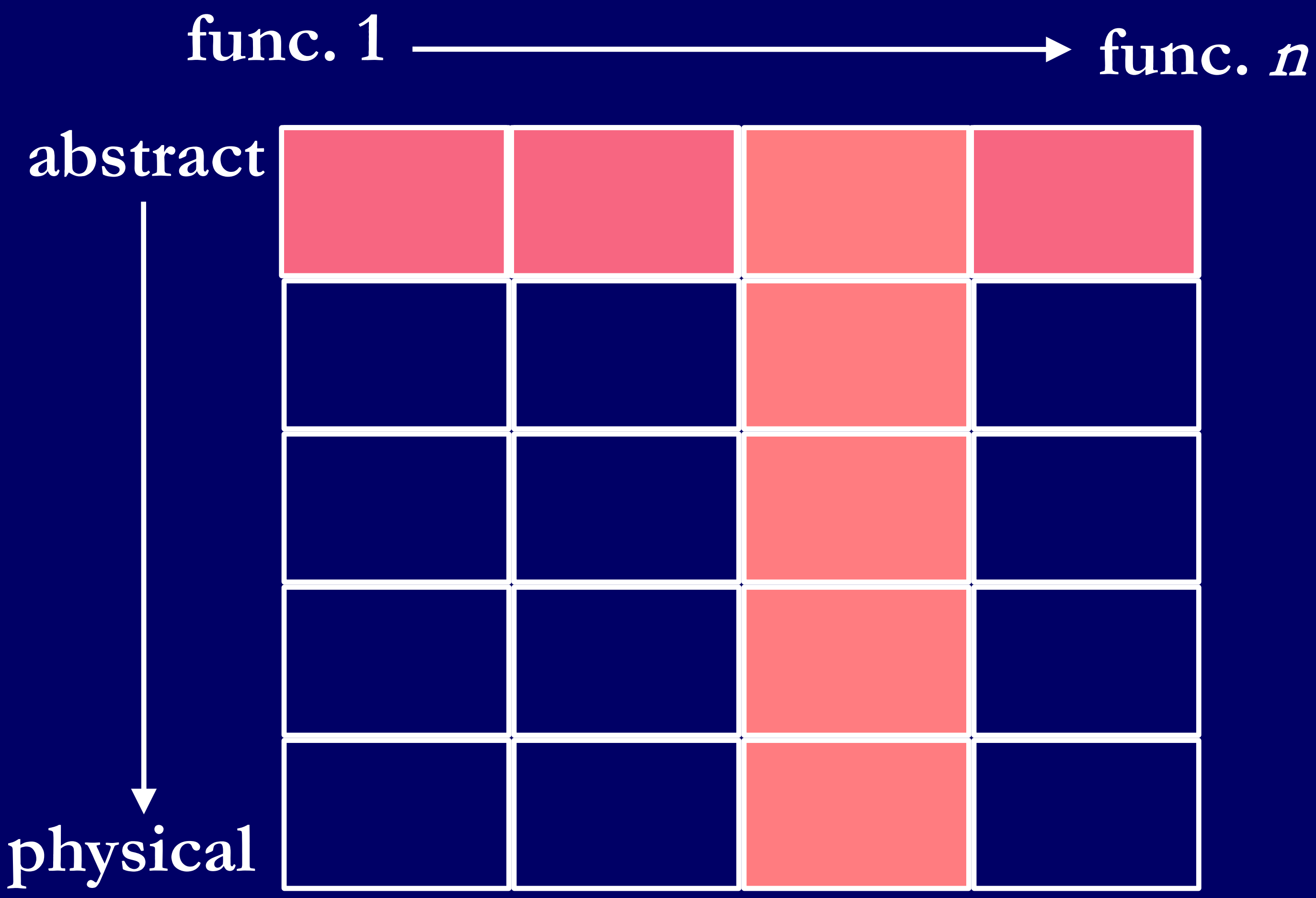
- Linear model
- "Rapid"
- Prototype



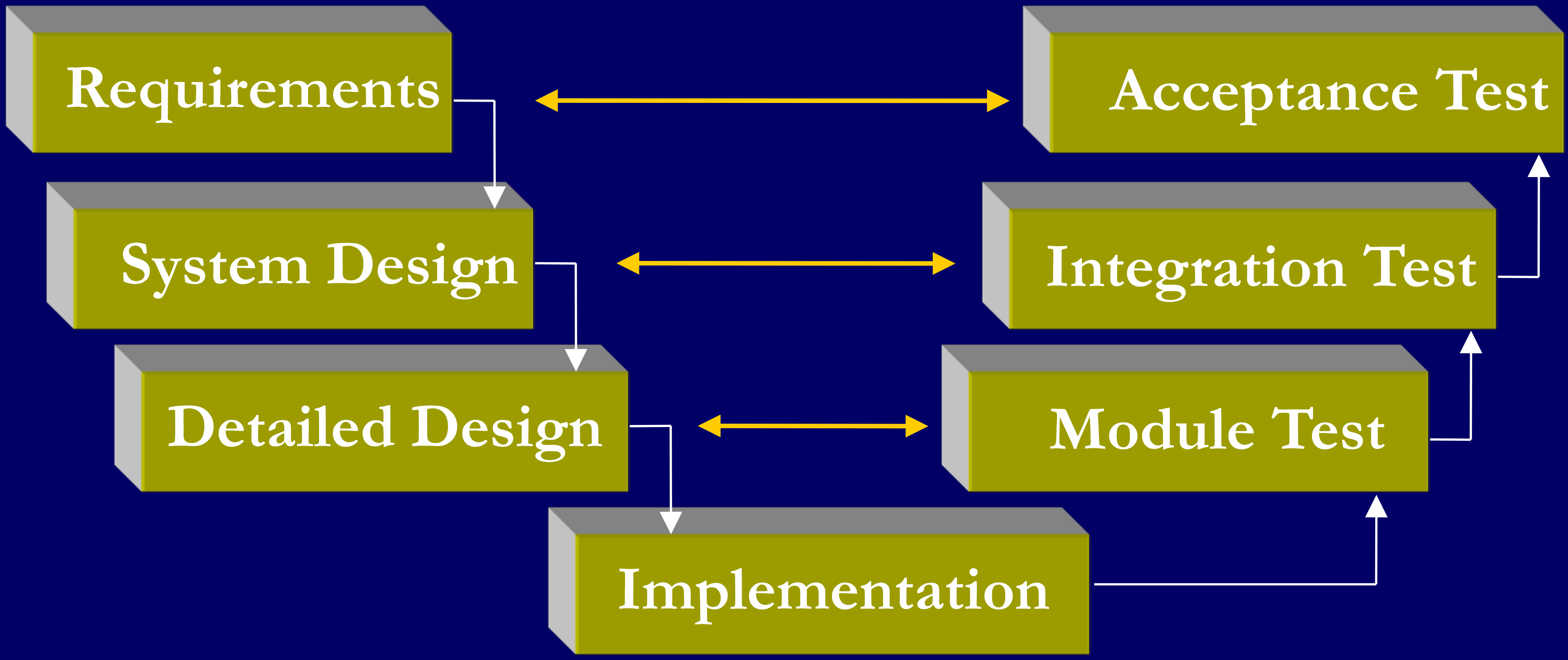
Horizontal Prototyping



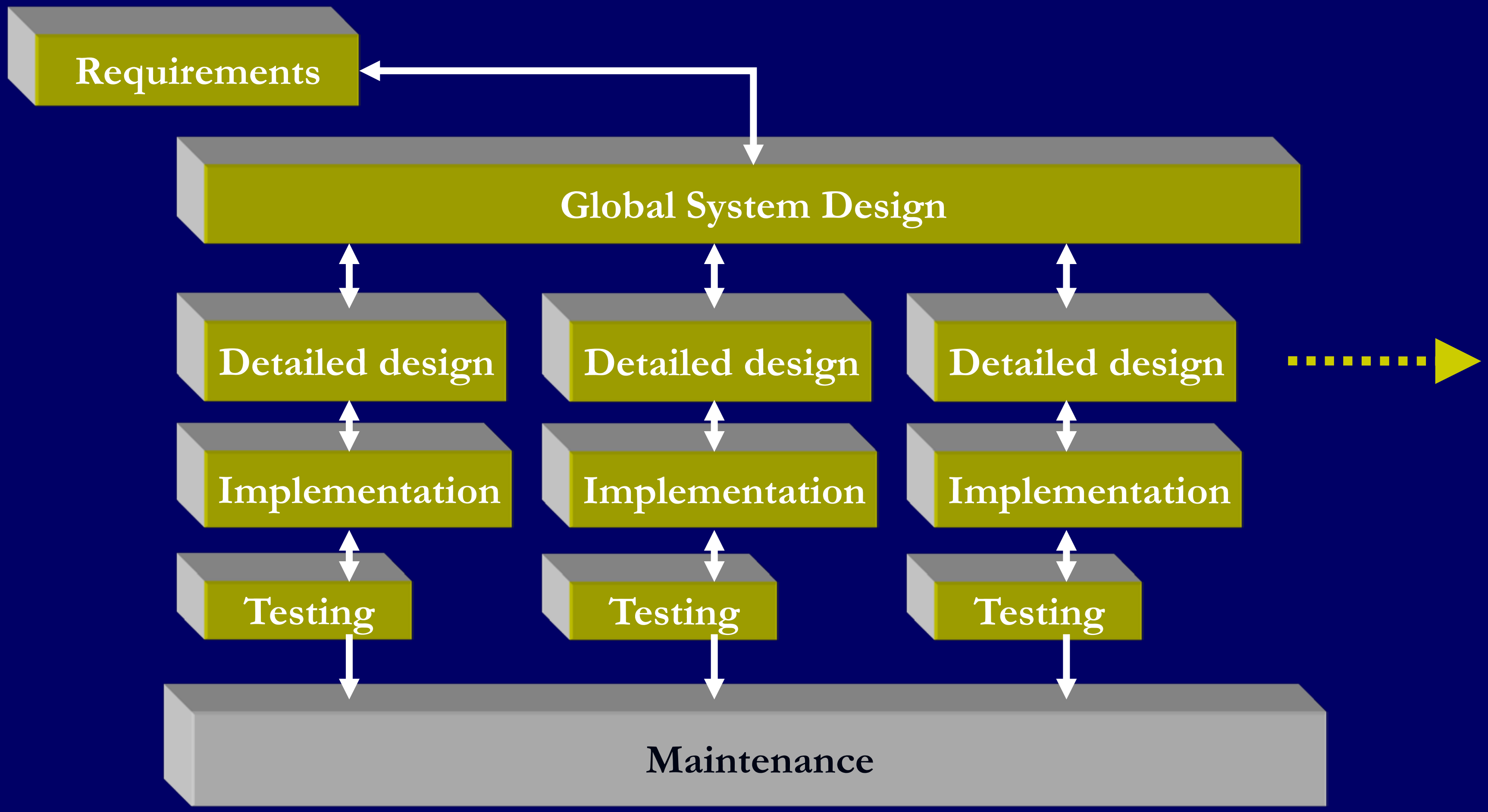
Vertical Prototyping



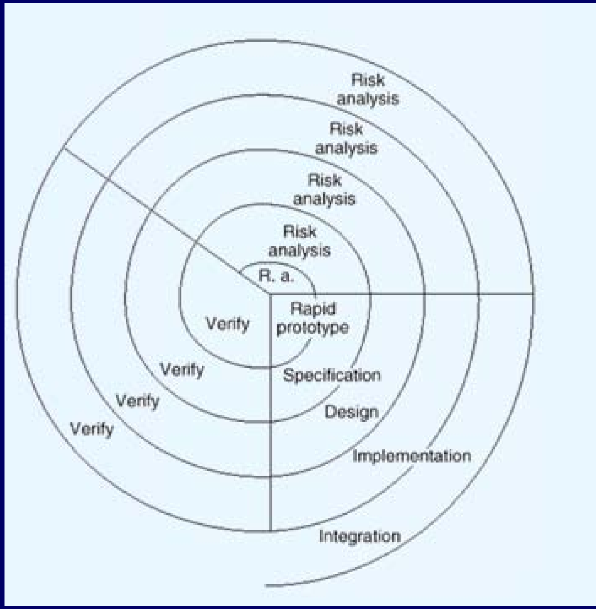
The V-Model


Requirements
Acceptance Test
System Design
Integration Test
Detailed Design
Module Test
Implementation

The Incremental Model



Spiral Model



Spiral Model

- Key idea
  - Always some risk in software development
    - People leave…
    - Other products not delivered on time
  - We need to minimize risk
    - e.g., building prototypes & simulations minimizes risks

Spiral Model

- Key idea
  - Always some risk in software development
    - People leave…
    - Other products not delivered on time
  - We need to minimize risk
    - e.g., building prototypes & simulations minimizes risks
- Precede each phase by
  - looking at alternatives
  - risk analysis

Spiral Model

- Key idea
  - Always some risk in software development
    - People leave…
    - Other products not delivered on time
  - We need to minimize risk
    - e.g., building prototypes & simulations minimizes risks
- Precede each phase by
  - looking at alternatives
  - risk analysis
- Follow each phase by
  - evaluation
  - planning of next phase

The Spiral Model

1. Build what you think you need
   - Perhaps using the waterfall model

The Spiral Model

1. Build what you think you need
   – Perhaps using the waterfall model
2. Get a few users to help you debug it!!
   – First an “alpha” release, then a “beta” release

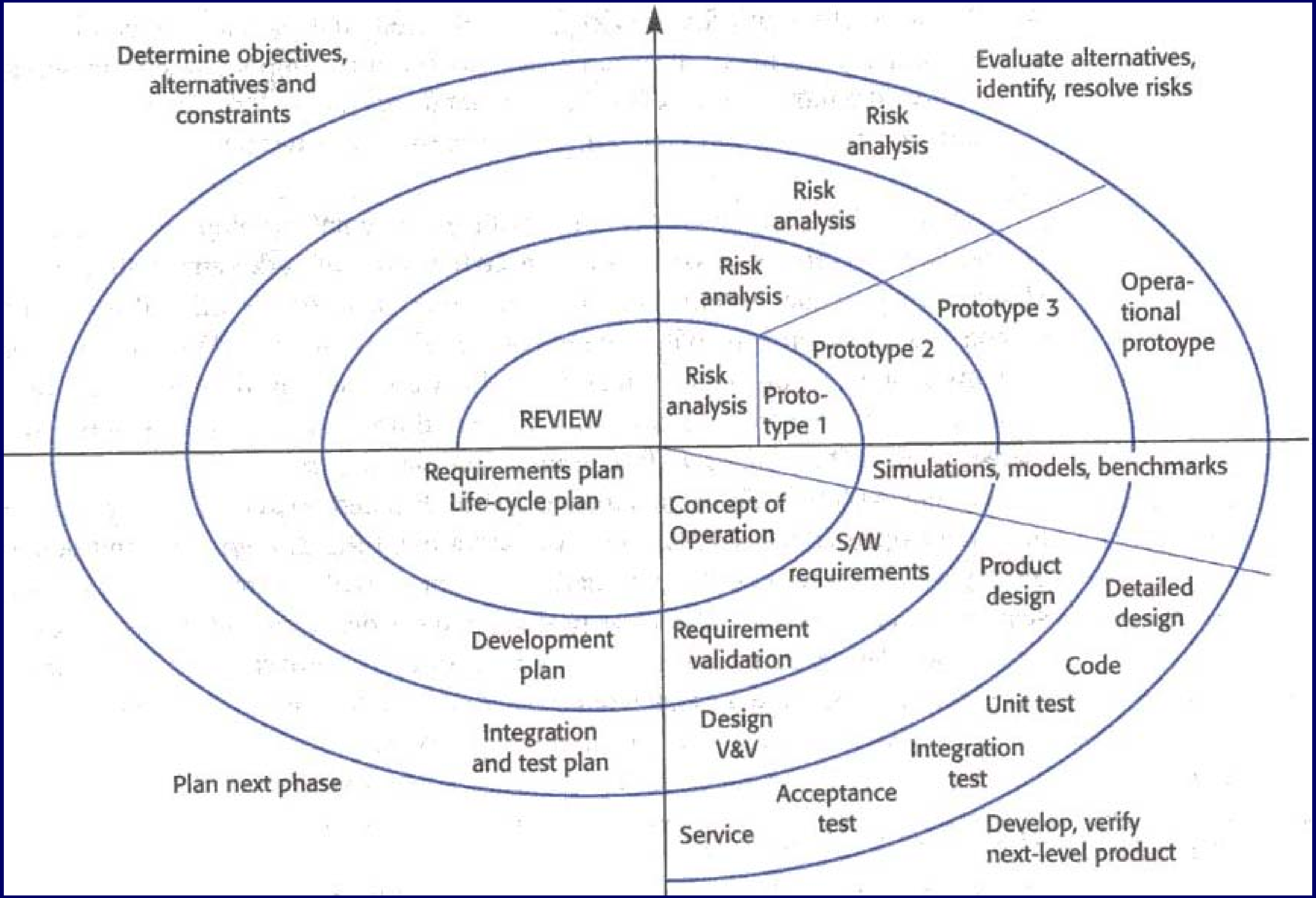
The Spiral Model

1. Build what you think you need
   – Perhaps using the waterfall model
2. Get a few users to help you debug it!!
   – First an “alpha” release, then a “beta” release
3. Release it as a product (version 1.0)
   – Make small changes as needed (1.1, 1.2, ….)

The Spiral Model

1. Build what you think you need
   - Perhaps using the waterfall model
2. Get a few users to help you debug it!!
   - First an “alpha” release, then a “beta” release
3. Release it as a product (version 1.0)
   - Make small changes as needed (1.1, 1.2, ….)
4. Save big changes for a major new release
   - Often based on a total redesign (2.0, 3.0, …)

The Spiral Model



Advantages of the Spiral Model

- Encourages prototyping
- Minimizes unnecessary elaborate specification
- Enables rework when needed
- Incorporates existing models
- Focuses on risk

Risk Assessment

- Risk-driven approach; in each spiral:
  - identify potential risks
  - plan next step based on risk analysis
  - refine design in highest-risk areas
- Explicitly attempts to identify potential problems
  - not just in initial stages of design
  - also later, when more has been learned about the problem and the design
- What are the “risky” parts of the system
  - relies on developer experience

Disadvantages of the Spiral Model

- Need good risk-assessment skills!

Risk Assessment Example

1) How many coders you have to work on the project.

Risk Assessment Example

1) How many coders you have to work on the project.
2) How many days each has till due date of next release (excluding vacations)?

Risk Assessment Example

1) How many coders you have to work on the project.
2) How many days each has till due date of next release (excluding vacations)?
3) How efficient is each coder?...

Risk Assessment Example

1) How many coders you have to work on the project.
2) How many days each has till due date of next release (excluding vacations)?
3) How efficient is each coder?...
4) How many features to be implemented in the coming release?

Risk Assessment Example

1) How many coders you have to work on the project.
2) How many days each has till due date of next release (excluding vacations)?
3) How efficient is each coder?...
4) How many features to be implemented in the coming release?
5) How many days each feature will take?

Risk Assessment Example

1) How many coders you have to work on the project.
2) How many days each has till due date of next release (excluding vacations)?
3) How efficient is each coder?...
4) How many features to be implemented in the coming release?
5) How many days each feature will take?
6) What’s the probability that you finish time?!

Capacity

Coder	Class	Days	Vacation	Work Factor		Effective Days
Philip	Manager	64	5 ± 1	0.1 ± 0.1		5.9 ± 3
Tracy	Architect	64	4	0.2 ± 0.1		12 ± 3.6
Sam	Lead	64	5 ± 1	0.4 ± 0.1		23.6 ± 5.9
Al	Lead	64	5	0.4 ± 0.1		23.6 ± 5.9
Goodwin	Lead	64	5	0.3 ± 0.1		17.7 ± 4.7
Christina	Coder	64	4	0.6 ± 0.1		36 ± 5.4
Shakur	Coder	64	0	1.2 ± 0		76.8 ± 0.6
Helen	Coder	64	5 ± 1	.6 ± 0.2		35.4 ± 11.8
Ted	Coder	50	4 ± 2	1 ± 0.4		46 ± 18.5
Cal	Coder	64	5 ± 1	0.6 ± 0.2		35.4 ± 11.8
Britney	Coder	60	3 ± 2	0.7 ± 0.1		39.9 ± 4.8
Bob	Coder	64	4 ± 1	0.7 ± 0.1		42 ± 4.9
totals:			49 ± 3.6	6.8 ± 0.5		394 ± 28

Requirements

fid	description	prereq	prio	promised	assigned	initial	status	to date	remain	spec	design
345	show angular separation		A	nto	al	4 ± 0.4	DONE	4		y	y
304	Field-of-View indicators		A	ttr	helen	8 ± 1	CC	7		y	y
234	NGC/IC objects		A	bpi	brit	22 ± 2	WIP	7	18 ± 1	y	y
389	time simulation		A	bpi, nto	sam, cal	35 ± 5	WIP	10	27 ± 4	y	y
230	change location		A		helen	5 ± 1	NYS		5 ± 1		
704	set location to city	230	A		bob	9 ± 0.5	WIP	7	3	y	y
298	set elevation	230	A		helen, cal	21 ± 4	NYS		21 ± 4	y	p
239	set location to planet	230	A		sam, brit	19 ± 2	NYS		19 ± 2	p	y
456	set location from map	230	A		bob	11 ± 0.5	NYS		11 ± 0.5	y	y
301	orbit framework		A		bob, al	32 ± 5	NYS		32 ± 5	y	y
303	orbit display	301	A		chris	14 ± 1	WIP	6	9 ± 2	p	y
906	orbit editor	301	A		shak	18 ± 2	WIP	12	8 ± 0.5	y	p
959	proper motion		A		phil, brit	15 ± 1	WIP	1	14 ± 1	y	y
508	selective constellations		A		tracy	4 ± 0.5	WIP	2	2	y	y
102	images for objects		A		shak	10 ± 0.5	NYS		10 ± 0.5		
294	show planets	102	A	nasa	ted	41 ± 3	WIP	9	27 ± 2	y	y
459	rendered planets	294	A	nasa	good	18 ± 5	WIP	3	17 ± 3	y	p
873	planet atmosphere	459	A	nasa	shak	8 ± 4	NYS		8 ± 4		y
939	custom images	102	A		shak	5 ± 0.2	NYS		5 ± 0.2	y	p
986	constellation boundaries		A			8 ± 1	NYS		8 ± 1	y	y
934	classical constellation		A		al	15 ± 3	NYS		15 ± 3	y	
904	clip movies		A		chris	22 ± 10	NYS		22 ± 10	y	
848	H-R diagram		A		helen	15 ± 1	NYS		15 ± 1	x	
509	night vision		A		shak	17 ± 5	NYS		17 ± 5	y	y
937	absolute motion		A			3 ± 0.3	NYS		3 ± 0.3	y	p
394	light pollution		A		shak	3 ± 0.3	NYS		3 ± 0.3	y	y
367	limit stars by distance		A		shak	15 ± 1.5	NYS		15 ± 1.5	y	y
totals (A)						397 ± 17	1/27	68	334 ± 14	78%	63%

Design Patterns

Pattern: Chain of Responsibility

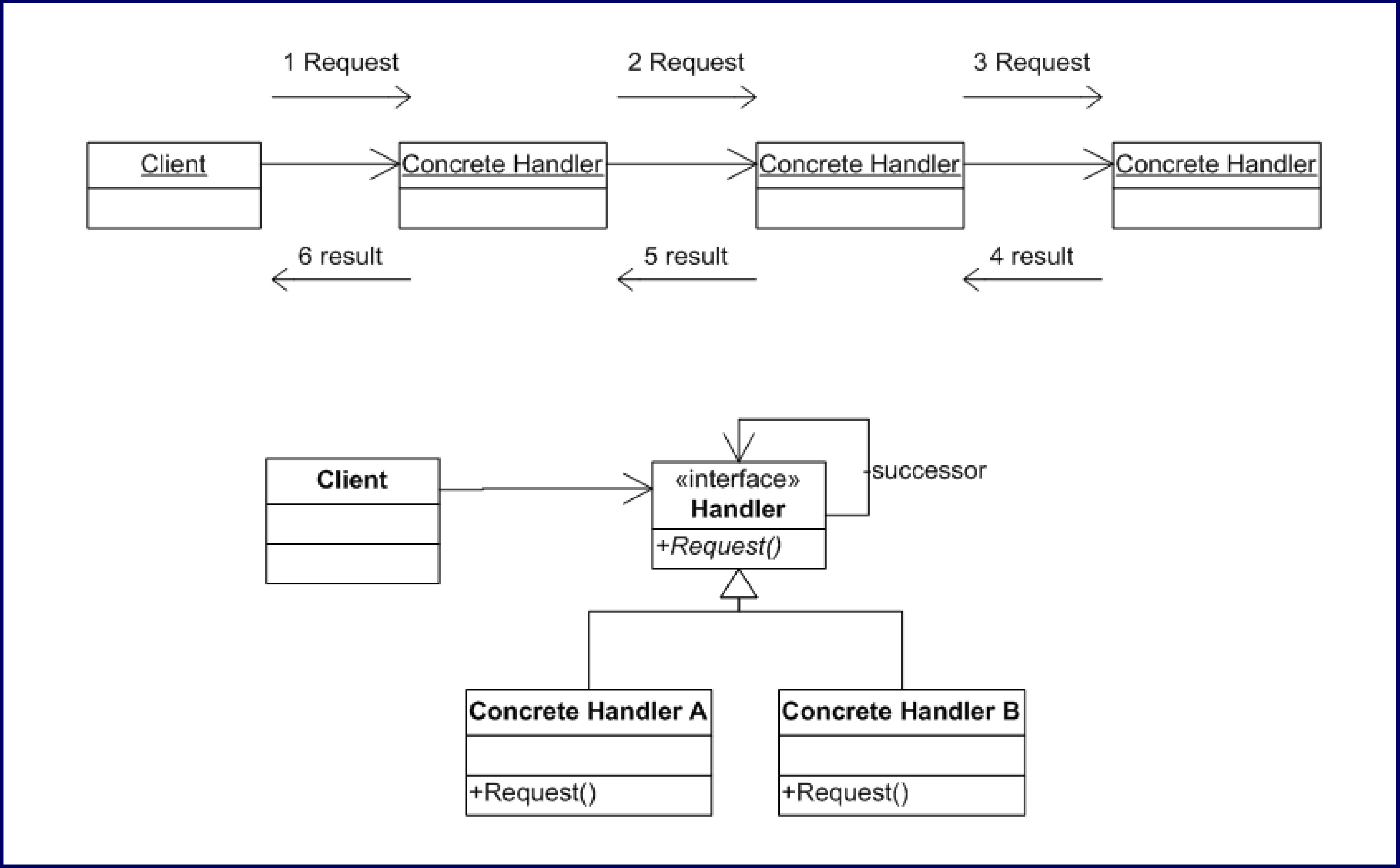
Problem

- Calling a method tightly couples the sending and receiving objects
  - Requires that you know what operation you want to perform, and which object you want to perform it
- This is usually not a problem, but sometimes …
  - You need an operation performed
  - There are multiple objects that can perform it
  - The object that should perform the operation changes over time and depends on the context
  - You might even want multiple objects to respond to the same request
- You need a way to call a method without specifically saying which object(s) should perform the operation

Solution

- Decouple sender and receiver by giving multiple objects a chance to handle a request
- When the sender needs to perform an operation, the request is passed along a chain of objects until one of them handles it
- The result is passed back along the chain to the sender
- The sender doesn't even know which object processed the request
- Any object that wants a chance to handle requests is added to the chain
- You can let multiple objects handle the same request by passing it all the way down the chain, even if somebody has already handled it

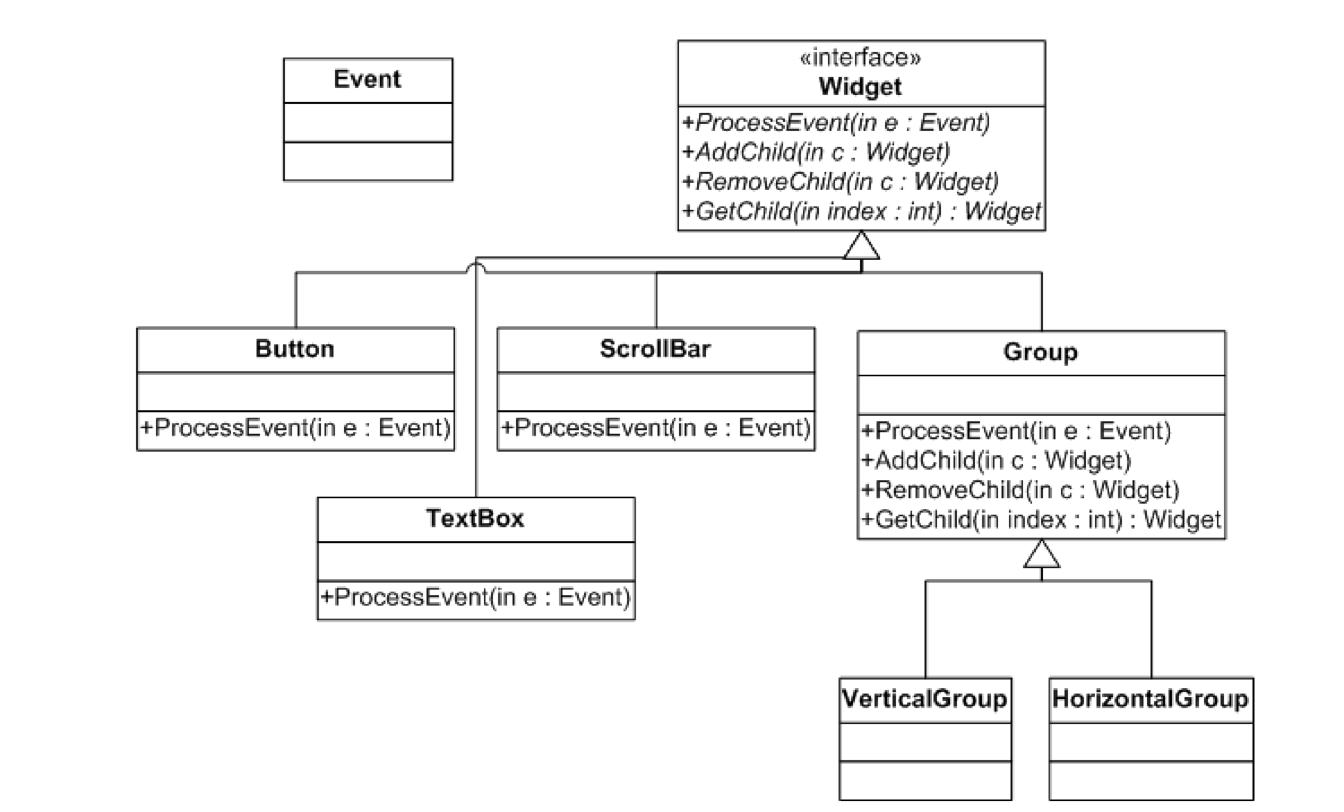
Solution: chain of responsibility



http://en.wikipedia.org/wiki/Chain-of-responsibility_pattern

Known Uses: UI Event Handling

- User interfaces are implemented as composites (ie, trees) of "widgets"



Known Uses: UI Event Handling

- Events are handled bottom-up
- The chain of responsibility includes the leaf-level widget where the event begins and its ancestors

- **Context-sensitive help can be handled the same way**
  - **ProcessHelp() instead of ProcessEvent()**

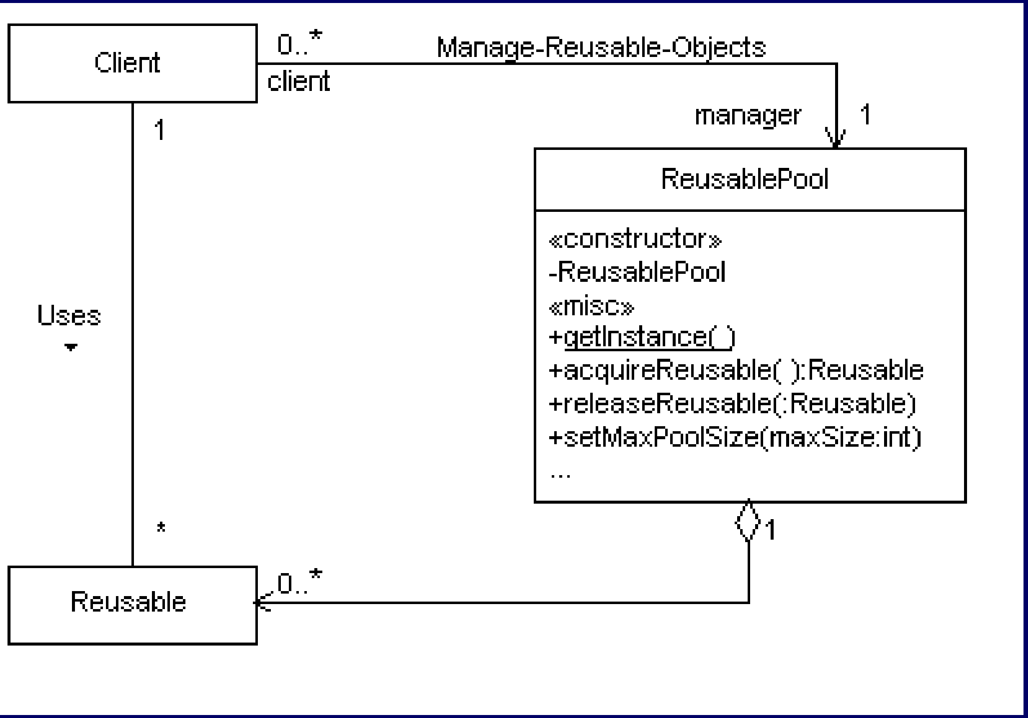
Pattern: object pool

A store of objects…

Problem

- Object creation is expensive
- Need to track how many an application create

Solution: object pool!



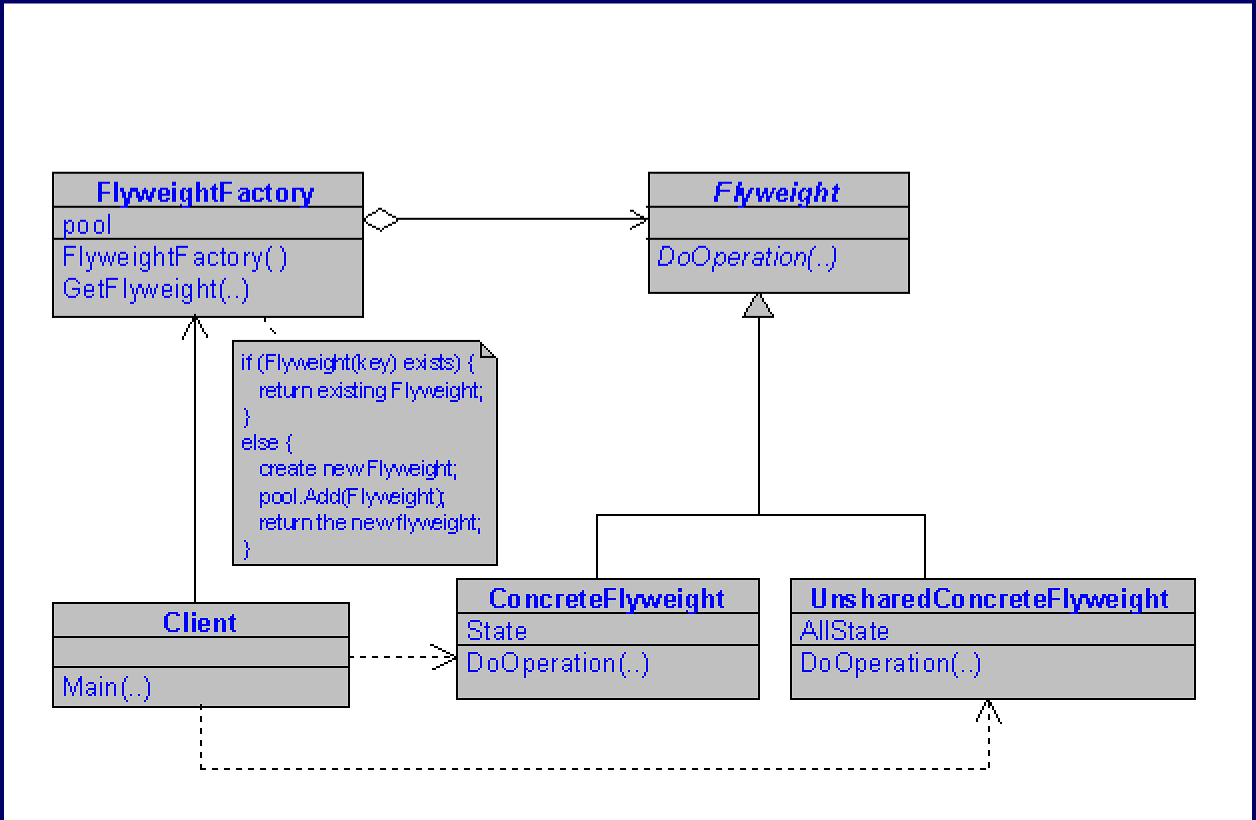
http://sourcemaking.com/design_patterns/object_pool/java

Pattern: flyweight

Problem

- Some objects take too much memory - can't afford to create many instances of it if the client code requests that…

Solution



http://en.wikipedia.org/wiki/Flyweight_pattern

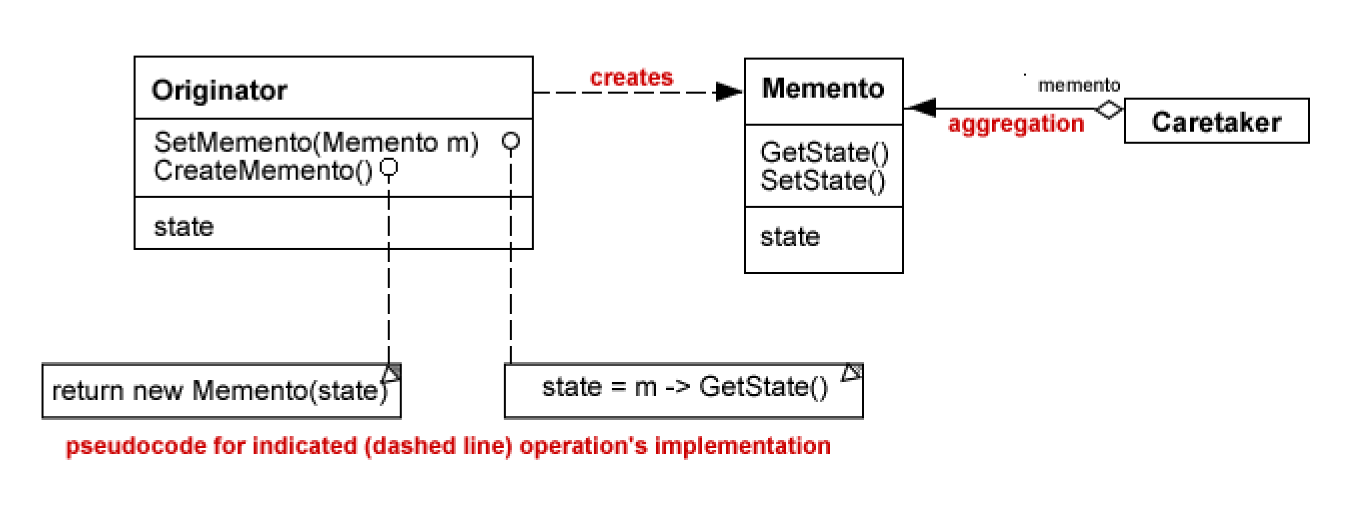
Pattern: memento

Problem

- What's the best way to save objects and restore them to/from harddisk...?

Solution

- Have each class responsible for it's saving and loading…



http://en.wikipedia.org/wiki/Memento_pattern

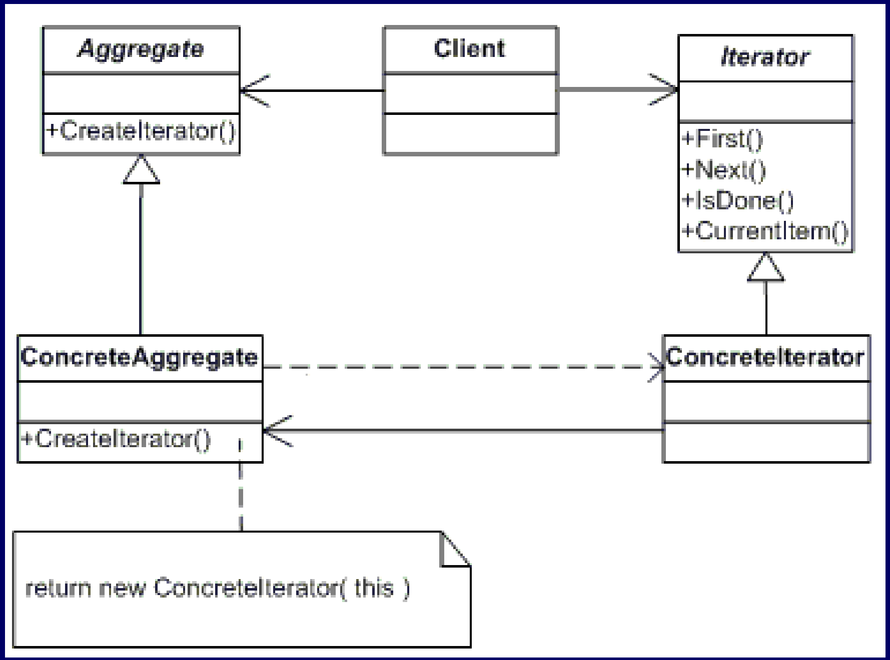
Pattern: iterator

Problem

- We need a standard way to work with *all* linear data structures, and similarly for trees, graphs, etc…

Solution

- Have each data structure implements the same interface



http://sourcemaking.com/design_patterns/iterator/java/1

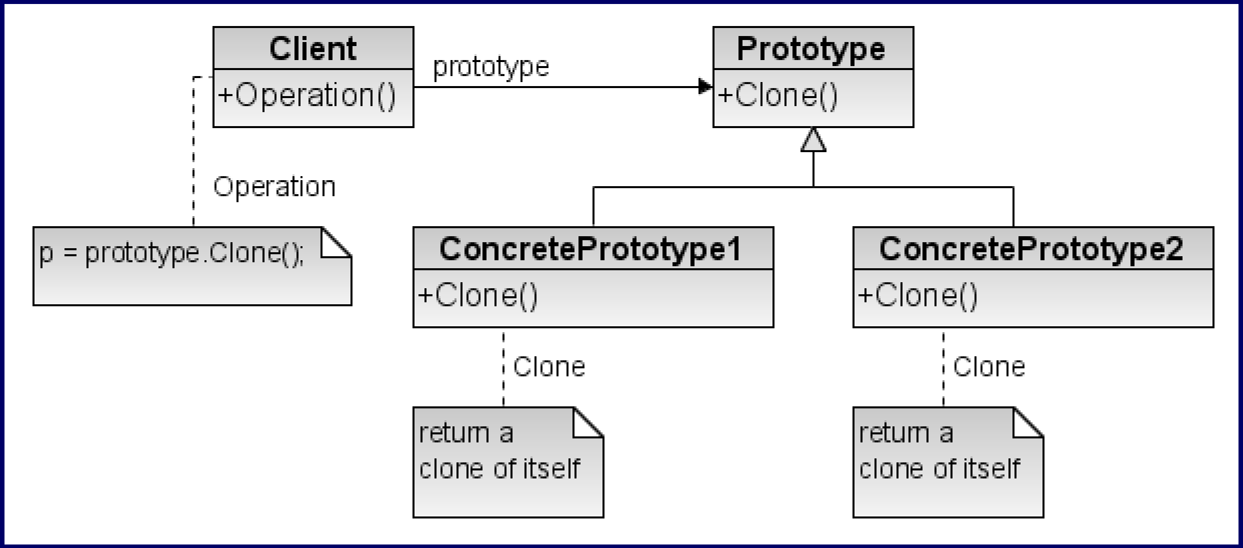
Pattern: prototype

Problem

- We need a way to create copies of object without knowing about the internal implementation of that object

Solution

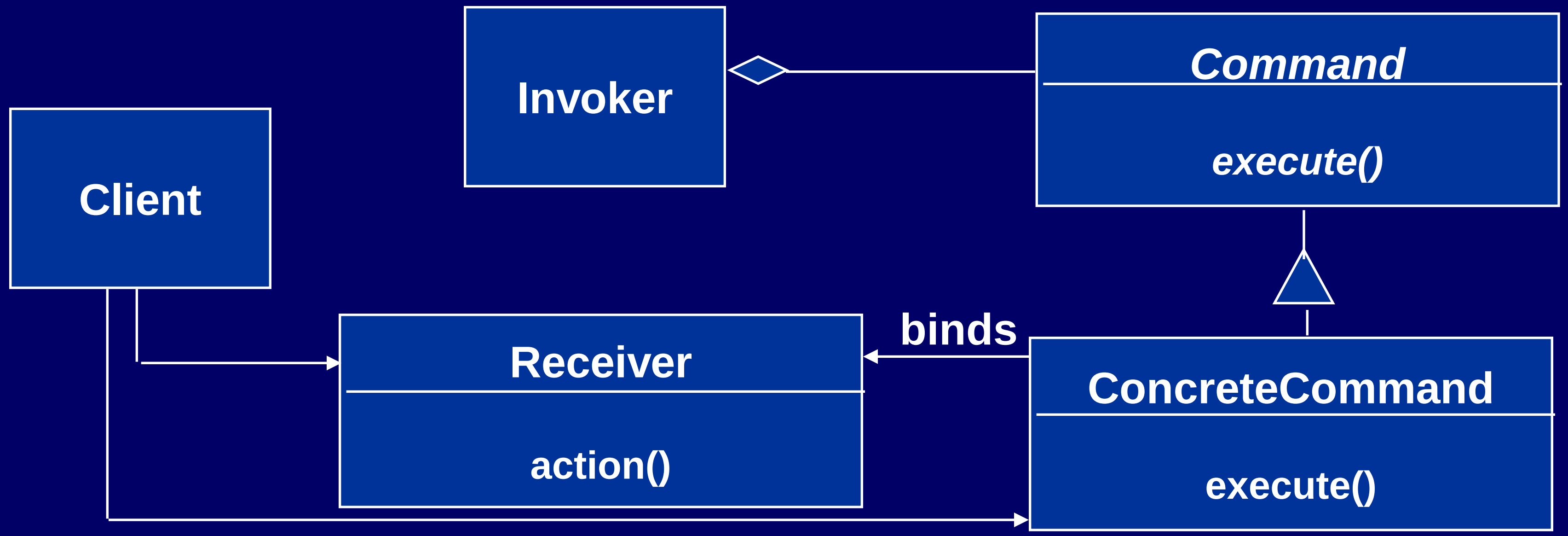
- Have each class clones it self, i.e. make it clonable



http://sourcemaking.com/design_patterns/prototype/java/1

Design Patterns: a quick overview

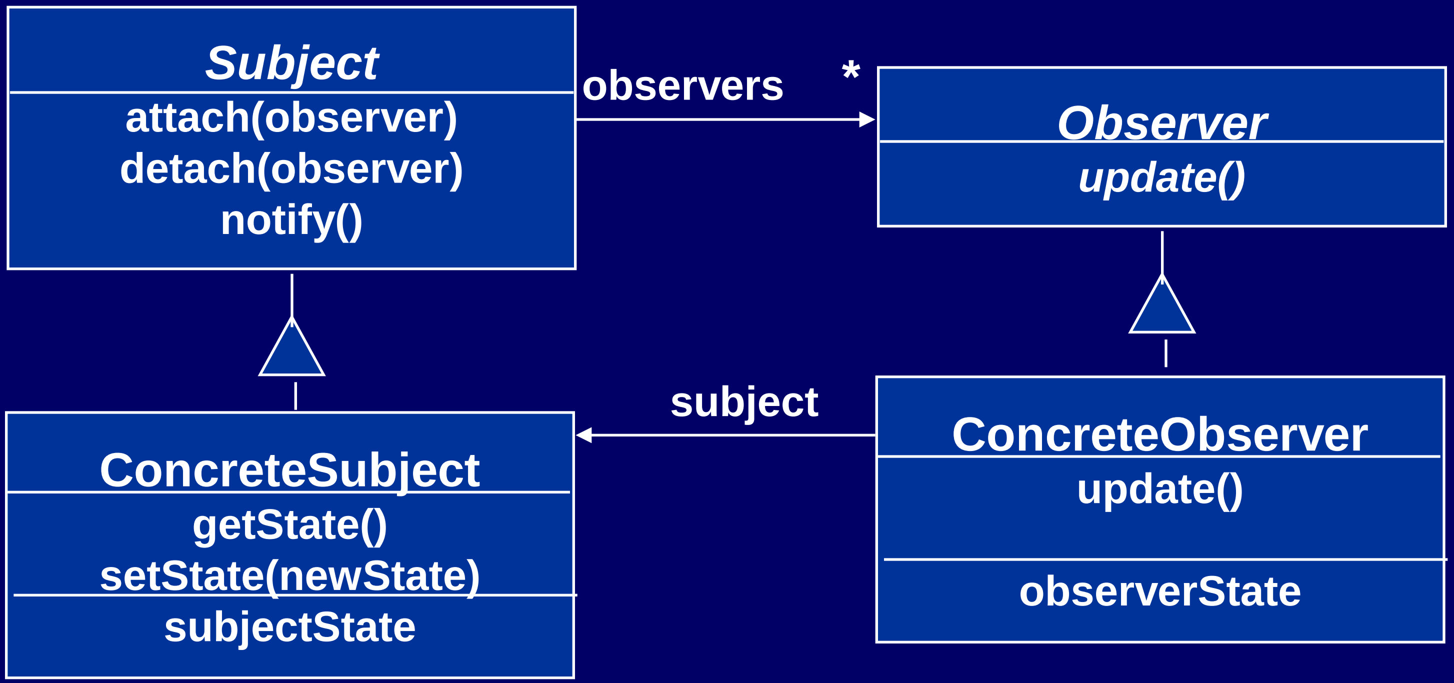
Command pattern



- Client creates a ConcreteCommand and binds it with a Receiver.
- Client hands the ConcreteCommand over to the Invoker which stores it.
- The Invoker has the responsibility to do the command (“execute” or “undo”).

http://sourcemaking.com/design_patterns/command/java/1

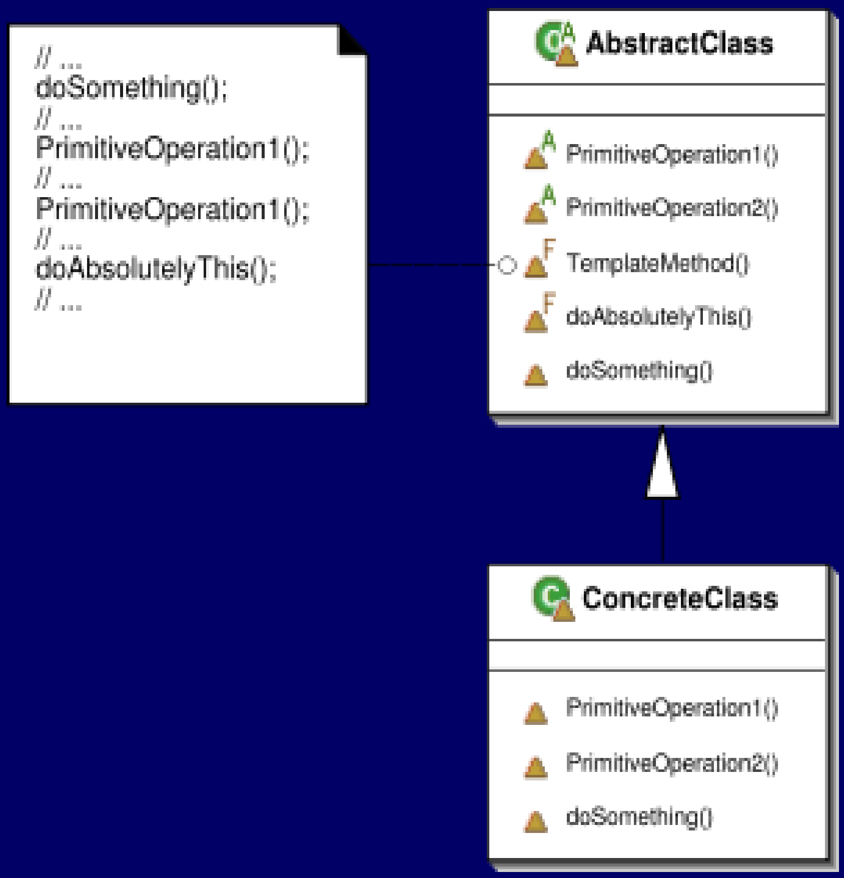
Observer pattern



- The Subject represents the actual state, the Observers represent different views of the state.
- Observer can be implemented as a Java interface.
- Subject is a super class (needs to store the observers vector)

http://sourcemaking.com/design_patterns/observer/java/1

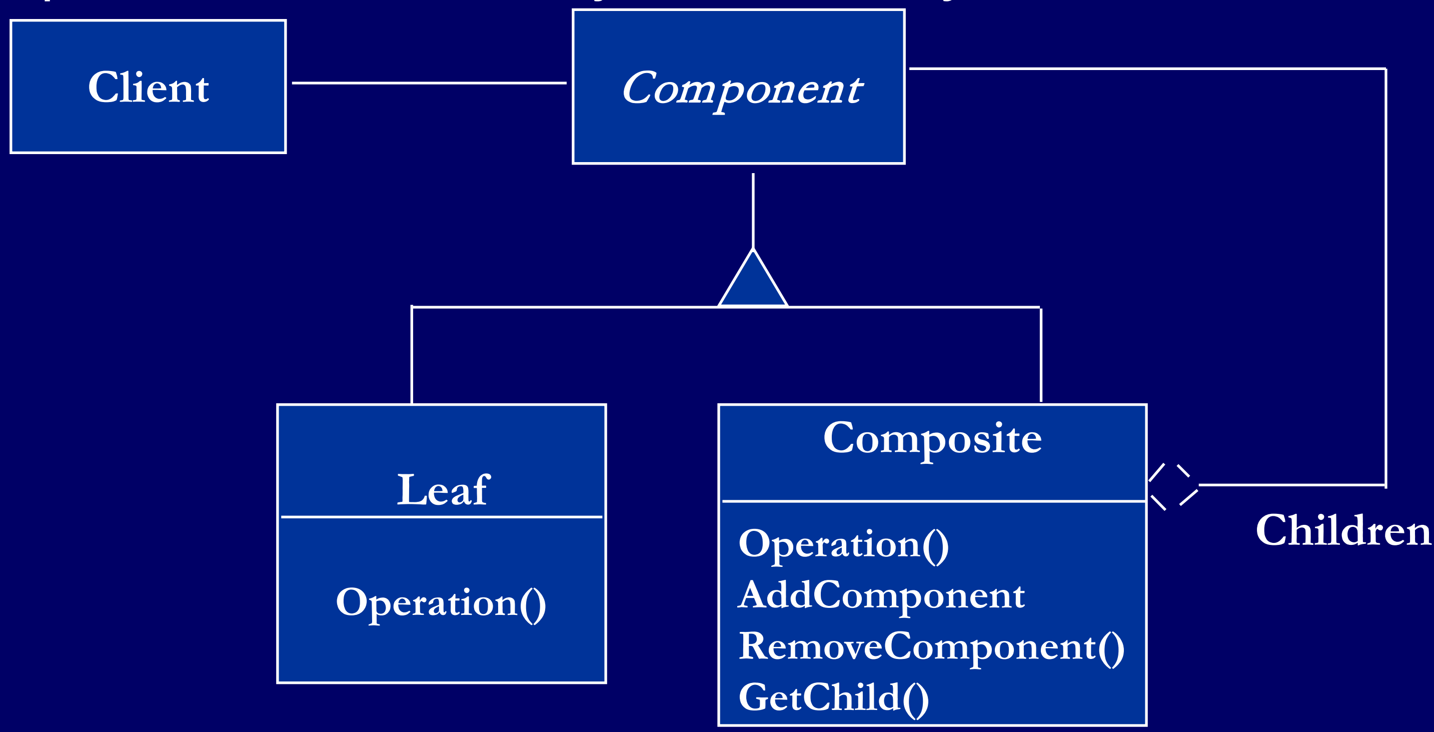
Template-method pattern



http://en.wikipedia.org/wiki/Template_method

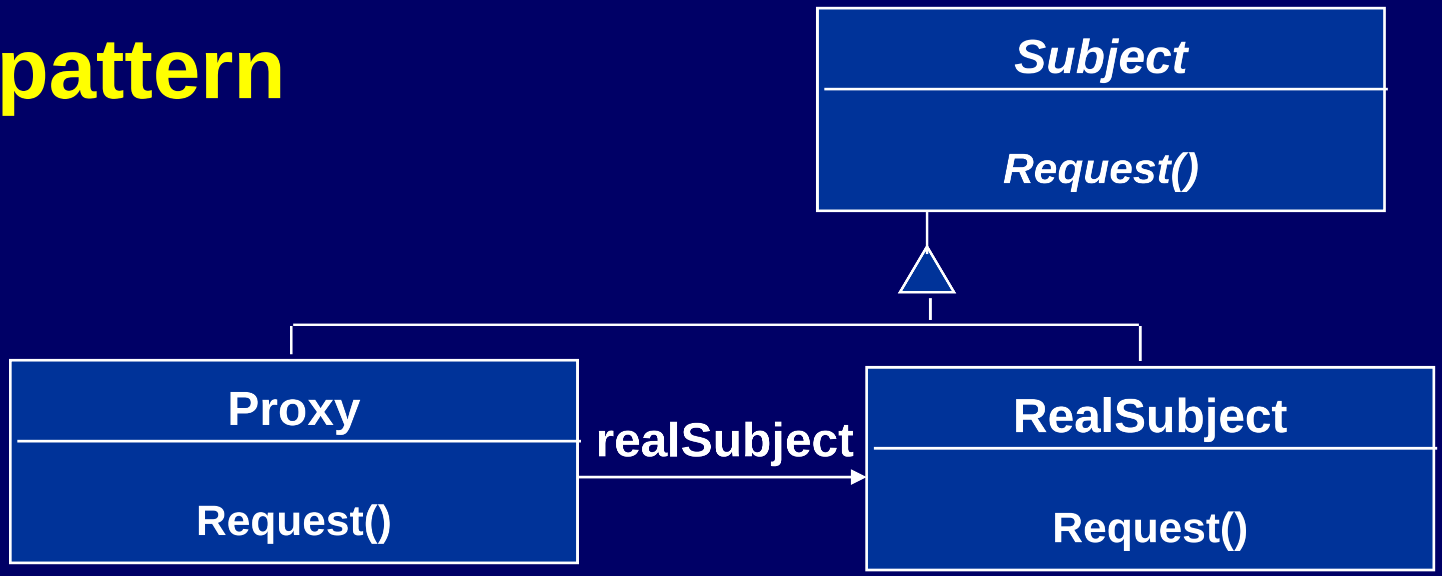
Composite Pattern

- Models tree structures that represent part-whole hierarchies with arbitrary depth and width.
- The Composite Pattern lets client treat individual objects and compositions of these objects uniformly



http://en.wikipedia.org/wiki/Composite_pattern

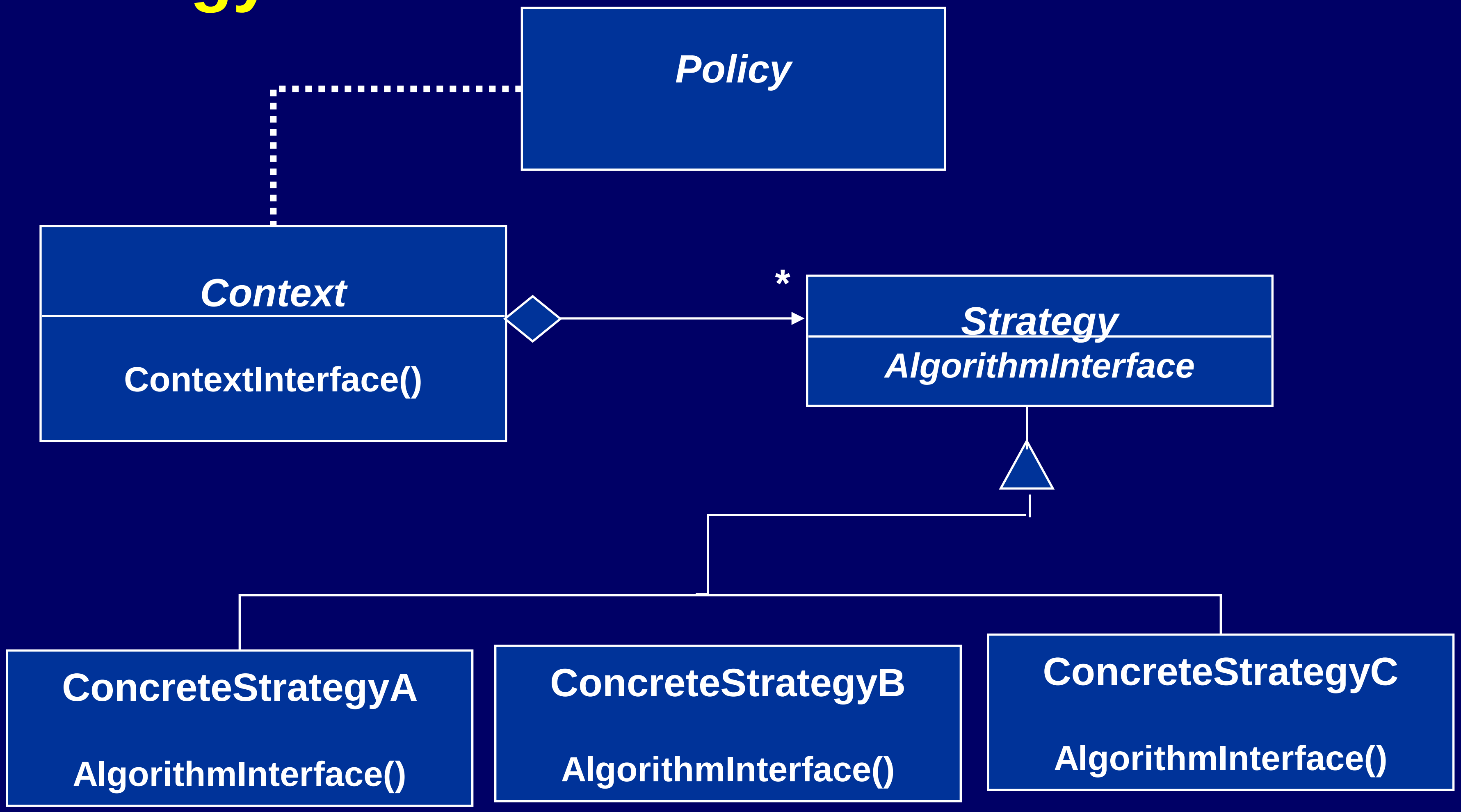
Proxy pattern



- Interface inheritance is used to specify the interface shared by Proxy and RealSubject.
- Delegation is used to catch and forward any accesses to the RealSubject (if desired)
- Proxy patterns can be used for lazy evaluation and for remote invocation.
- Proxy patterns can be implemented with a Java interface.

http://en.wikipedia.org/wiki/Proxy_pattern

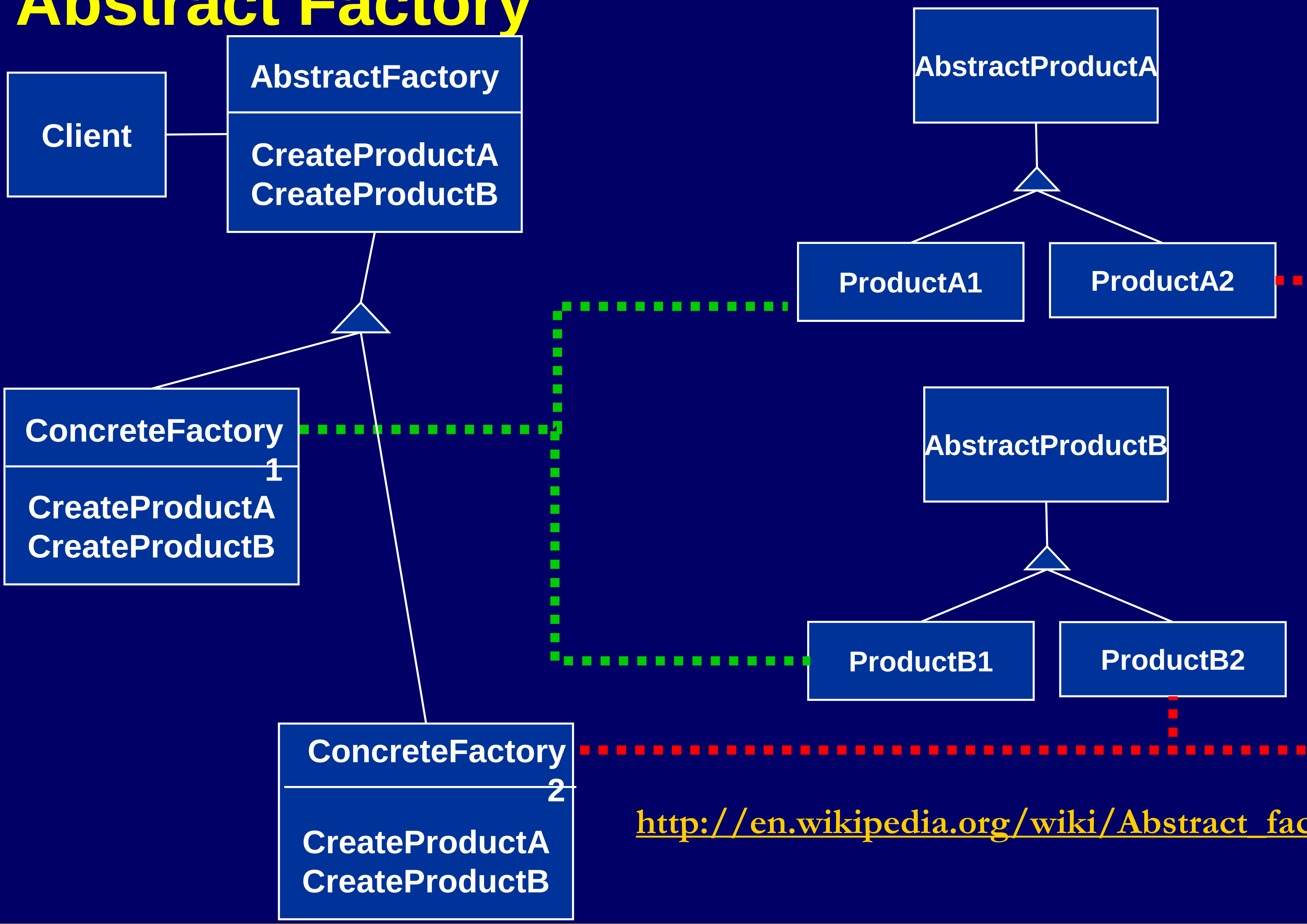
Strategy Pattern



Policy decides which Strategy is best given the current Context

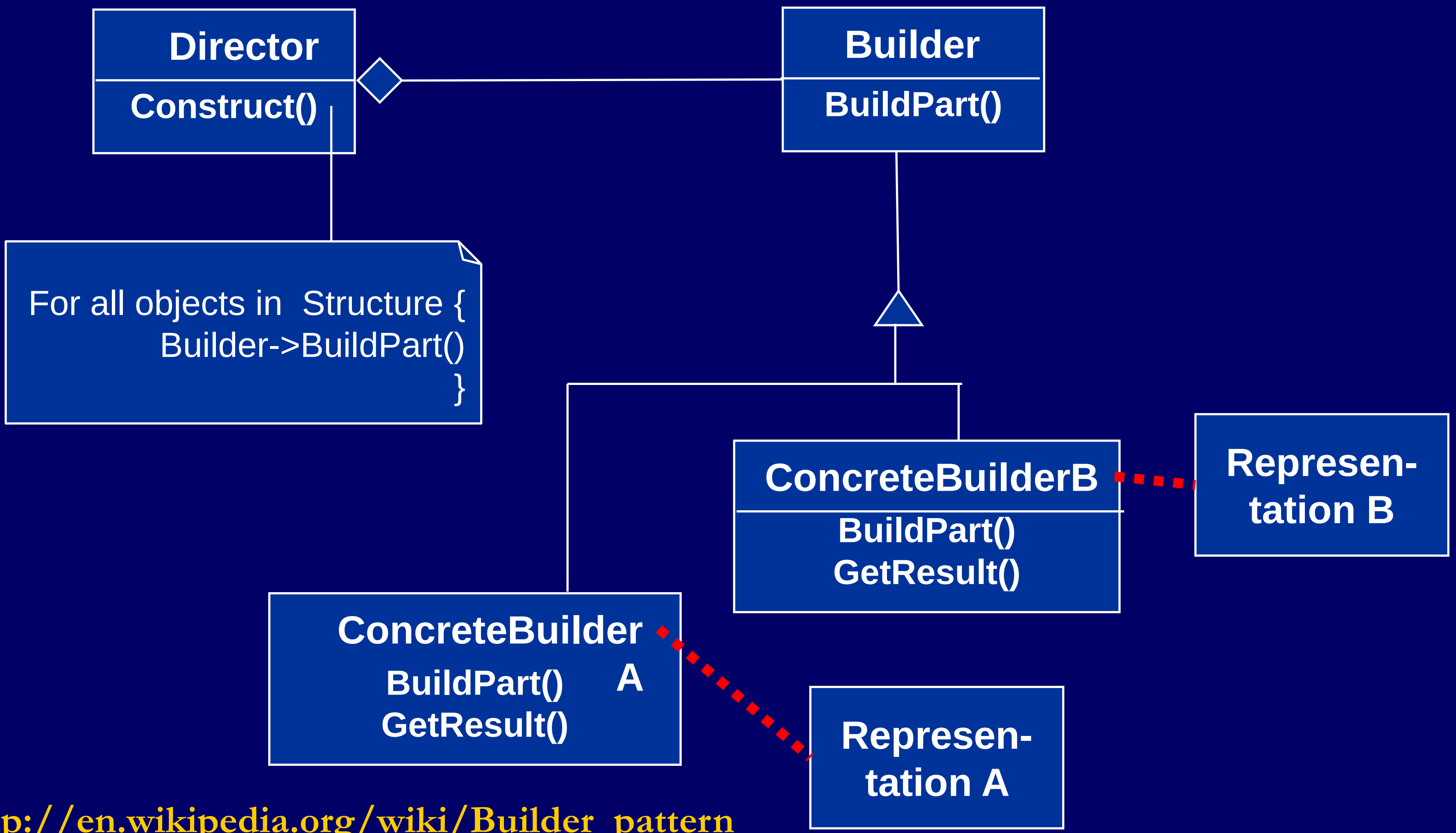
http://en.wikipedia.org/wiki/Strategy_pattern

Abstract Factory



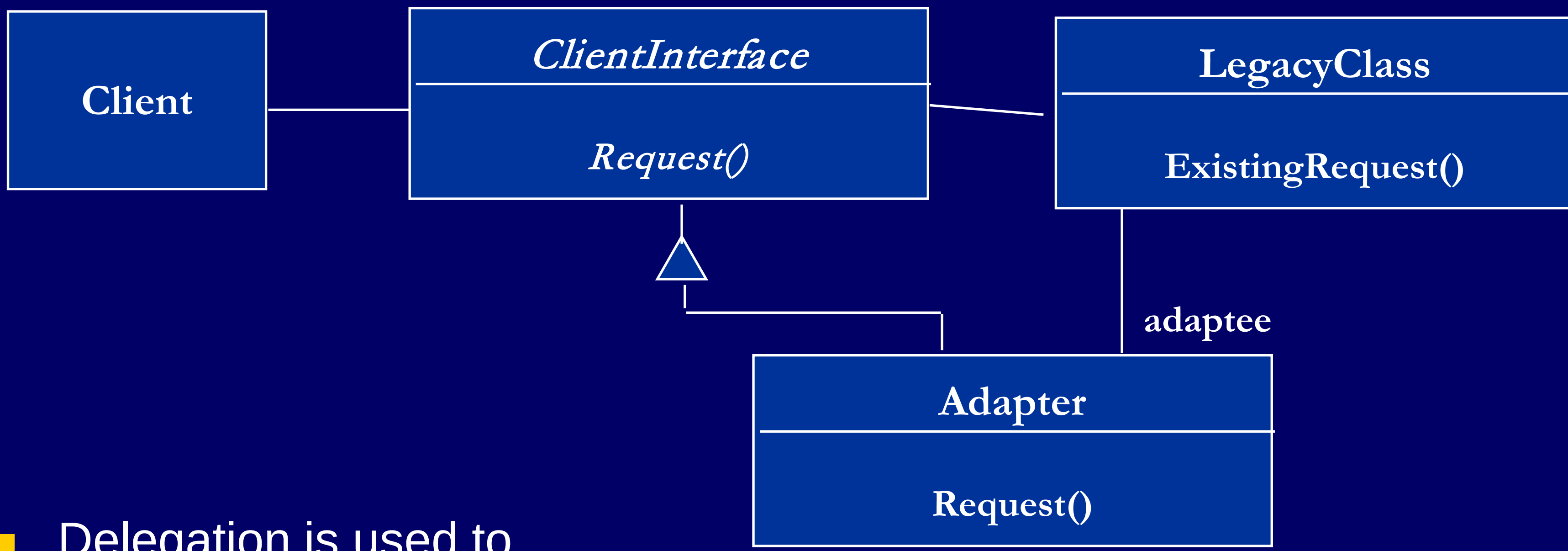
http://en.wikipedia.org/wiki/Abstract_factory

Builder Pattern



http://en.wikipedia.org/wiki/Builder_pattern

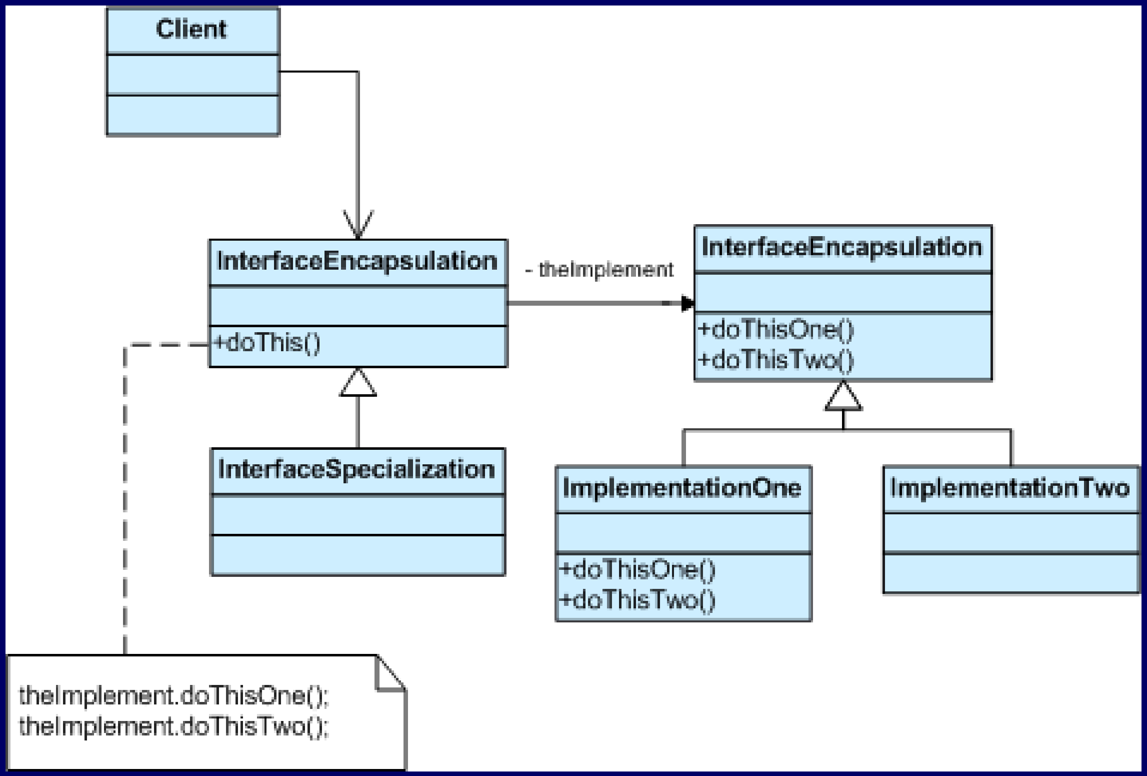
Adapter pattern



- Delegation is used to bind an **Adapter** and an **Adaptee**
- Interface inheritance is use to specify the interface of the **Adapter** class.
- ***Target*** and **Adaptee** (usually called legacy system) pre-exist the **Adapter.**
- **Target** may be realized as an interface in Java.

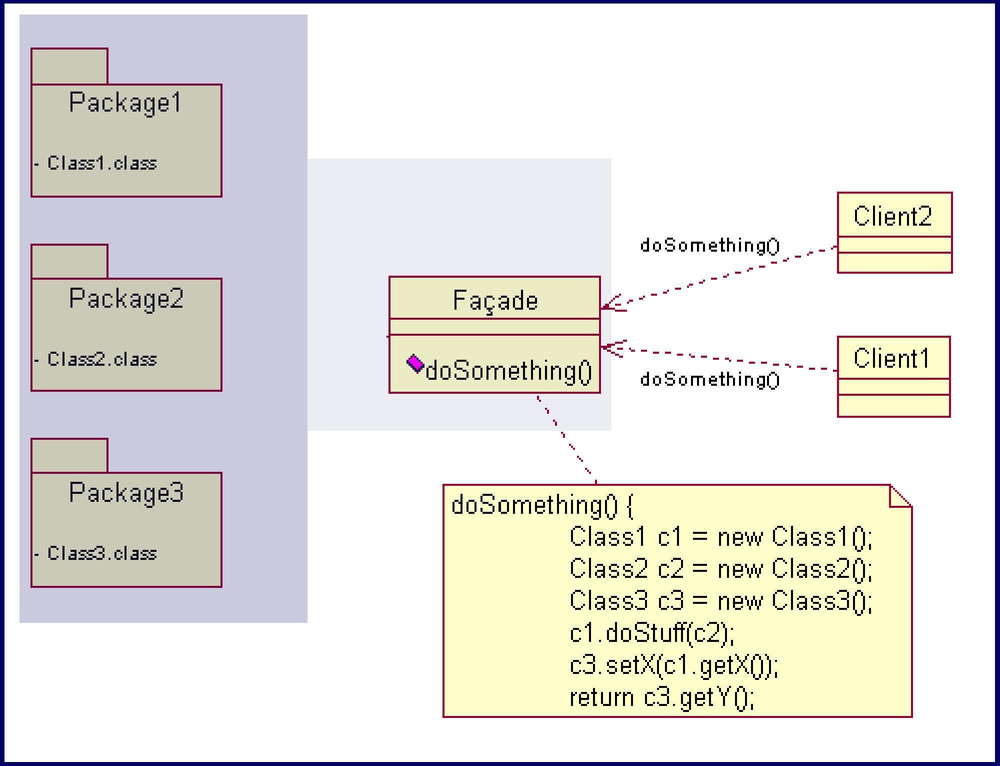
http://sourcemaking.com/design_patterns/adapter/java/1

Bridge Pattern

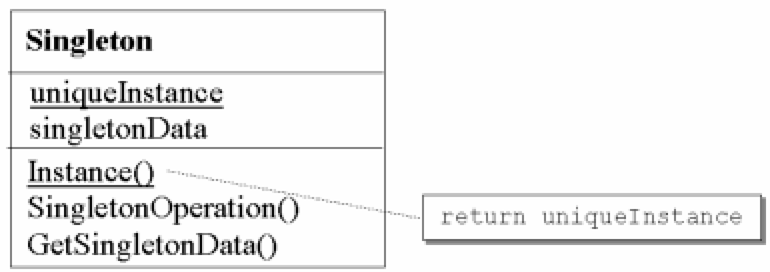


http://en.wikipedia.org/wiki/Bridge_pattern

Facade Pattern



Singleton Pattern



http://sourcemaking.com/design_patterns/singleton/java/1

Design Example