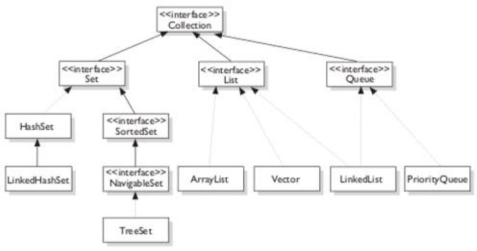
Lambda Expressions

```java
public interface FunctionalInterface{
        String func(String str);
}
```

```java
FunctionalInterface f;
f=(String str)->{return new StringBuilder(str).reverse().toString();};

System.out.println(f.func("Hello World!"));
```

Methods of collections <E>

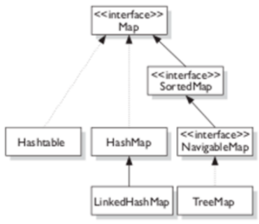
boolean add(E e)
boolean addAll(Collection<? Extends E> c)
void clear()
boolean contains(Object o)
boolean containsAll(Collection<?> c)
boolean isEmpty()
Iterator<E> iterator()
boolean remove(Object o)
boolean removeAll(Collection<?> c)
boolean retainAll(Collection<?> c)
int size()
Object[] toArray()
<T> T[] toArray(T[] a)



Example:
```java
Set<String> names=new HashSet<String>();
//…
List<String> members=new ArrayList<String>();
members.add("Moshe");
members.addAll(names);
```

Methods of maps <K,V>

V put(K **key**, V **value**)
void putAll(Map<? extends K, ? extends V > m)
V get(K **key**)
void clear()
boolean containsKey(Object **key**)
boolean containsValue(Object **value**)
boolean isEmpty()
V remove(Object **key**)
int size()
Collection<V> values()
Set<K> keySet()
Set<Map.Entry<K,V>> entrySet()



Example:
```java
Map<Integer, Employee> workers;
workers=new HashMap<Integer, Employee>();
workers.put(123456789, new Employee());
```

```java
interface Comparator <T> {
      int compare(T t1, T t2);
}
```

```java
interface Comparable <T> {
      int compareTo(T t) ;
}
```

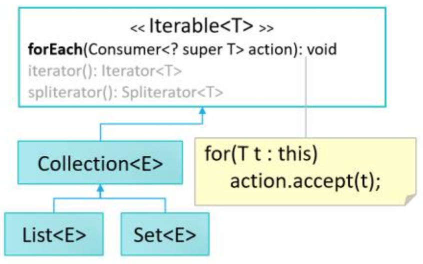
```java
for(Worker w : workers)
 System.out.println(w);
```

It is actually a shortcut for an ***Iterator***

```java
Iterator<Worker> it=workers.iterator();
while(it.hasNext())
 System.out.println(it.next());
```

ForEach

```java
List<Integer> list=Arrays.asList(10,12,35);

Consumer<? super Integer> action = new Consumer<Integer>() {

        @Override
        public void accept(Integer i) {
                System.out.println(i);
        }
};

list.forEach(action);

list.forEach(i->System.out.println(i));

list.forEach(System.out::println);
```



And many more interfaces & classes…

Common Java8 Functional Interfaces

Predicate<T>	- tests the T
Consumer<T>	- applies an action on the T
Function<T,U>	- given a T, returns a U (transformation)
BiFunction<T,U,V>	- transforms (T,U) into a V
Supplier<T>	- provides an instance of a T
UnaryOperator<T>	- a unary operator T → T
BinaryOperator<T>	- a binary oprator (T,T) → T

java.util.function.*

```java
public void fillDetailsForm(){
 String email="abc.gmail.com";
 try {
       pd.setEmail(email);
       System.out.println("this will not be printed");
 } catch (Exception e) {
       System.out.println("catching...");
       return; // exit the method

 } finally{

 }
 // and the code will not continue here...
 System.out.println("this will not be printed");
}
```

Buffered Reader/Writer Example

```
BufferedReader reader = null;
PrintWriter writer = null;
reader = new BufferedReader(new FileReader("in.txt"));
writer = new PrintWriter(new FileWriter("out.txt"));

String line;
while ((line = reader.readLine()) != null) {
        writer.println(line);
}

reader.close();
writer.close();
```

```
String input="1 fish 2 fish red fish blue";
Scanner s=new Scanner(input);
s.useDelimiter(" fish ");
System.out.println(s.nextInt());
System.out.println(s.nextInt());
System.out.println(s.next());
System.out.println(s.next());
```

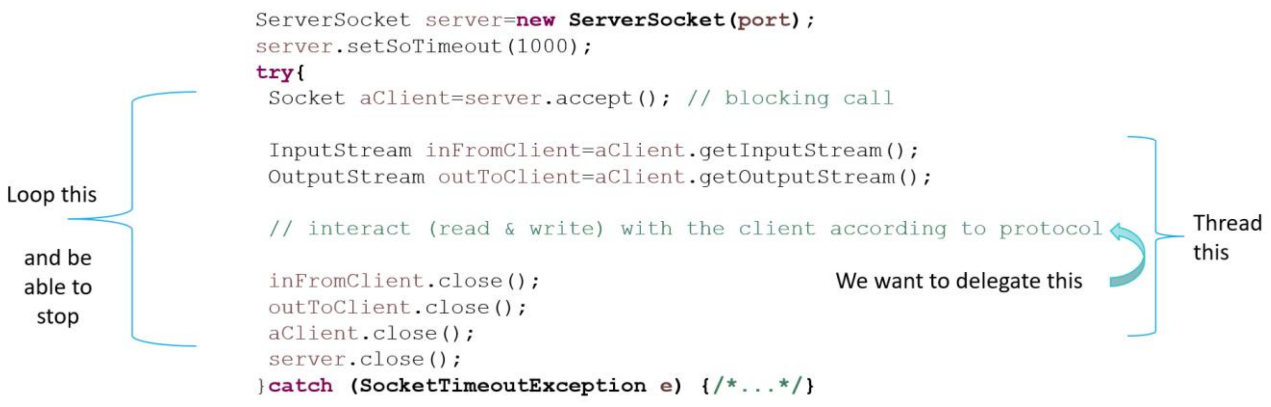
```
BufferedReader in = new BufferedReader(
      new InputStreamReader(System.in));
String line = in.readLine();
```

לקוח

```java
public void start(String ip, int port){
  try {
    Socket theServer=new Socket(ip, port);
    System.out.println("connected to server");

    BufferedReader userInput=new BufferedReader(new InputStreamReader(System.in));
    BufferedReader serverInput=new BufferedReader(new
                                   InputStreamReader(theServer.getInputStream()));

    PrintWriter outToServer=new PrintWriter(theServer.getOutputStream());
    PrintWriter outToScreen=new PrintWriter(System.out);

    // correspond according to a well-defined protocol
    readInputsAndSend(userInput,outToServer,"exit");
    readInputsAndSend(serverInput,outToScreen,"bye");

    userInput.close();
    serverInput.close();
    outToServer.close();
    outToScreen.close();
    theServer.close();

  } catch (UnknownHostException e) {/*...*/}
    catch (IOException e) {/*...*/}
}
```

```java
public static void main(String[] args) {
  String ip=args[0];
  int port = Integer.parseInt(args[1]);
  CLIclient client=new CLIclient();
  client.start(ip, port);
}
```

שרת



```java
ServerSocket server=new ServerSocket(port);
server.setSoTimeout(1000);
try{
 Socket aClient=server.accept(); // blocking call

 InputStream inFromClient=aClient.getInputStream();
 OutputStream outToClient=aClient.getOutputStream();

 // interact (read & write) with the client according to protocol

 inFromClient.close();
 outToClient.close();
 aClient.close();
 server.close();
}catch (SocketTimeoutException e) {/*...*/}
```

Stream

INTERMEDIATE	TERMINAL
returns a Stream	returns a result
distinct()	collect()
map()	count()
flatMap()	forEach()
limit()	min() , max()
peek()	reduce()
sorted()	toArray()
	findAny() , findFirst()
	allMatch() , andMatch() , noneMatch()

```
List<String> strings=Arrays.asList("the", "answer", "to", "life", "the", "universe",
"and", "everything", "=", "42");

int totalLength = strings.stream().map(String::length).reduce(0, (x,y)->x+y);
System.out.println(totalLength); // wow! its 42!!
```

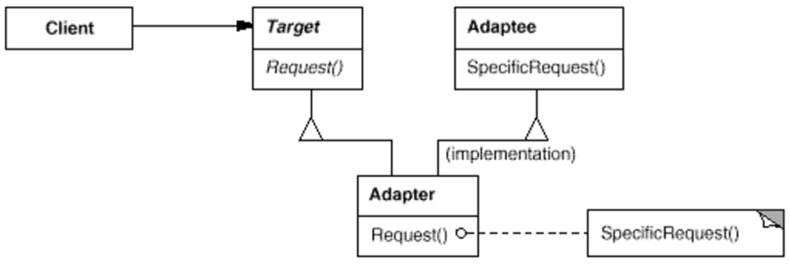
groupinBy

https://docs.oracle.com/javase/8/docs/api/java/util/stream/Collectors.html

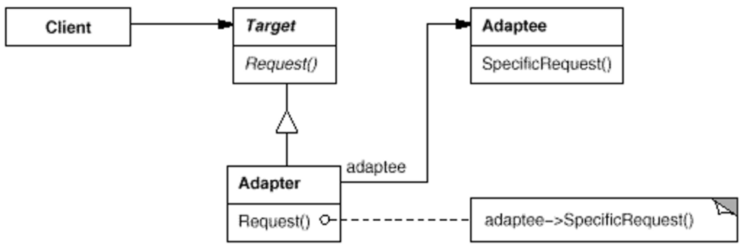
```
List<Employee> employees=new LinkedList<>();
employees.add(new Employee(18, "Moshe"));
employees.add(new Employee(18, "Tzipi"));
employees.add(new Employee(25, "Alon"));
employees.add(new Employee(22, "Tal"));
employees.add(new Employee(22, "Tomer"));

Map<Integer,List<Employee>> EmpByAge = employees.stream()
        .filter(e->e.name.startsWith("T"))
        .collect(Collectors.groupingBy(e->e.age));

EmpByAge.forEach((age,emps)->{
        System.out.println(age+":");
        emps.forEach(e->System.out.println("\t"+e.name));
});
```

```
output:
18:
        Tzipi
22:
        Tal
        Tomer
```

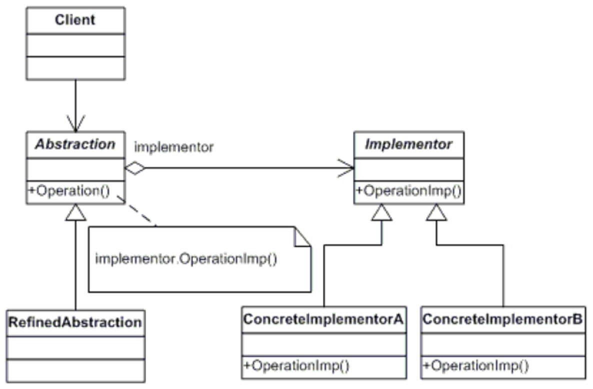
Class Adapter Pattern



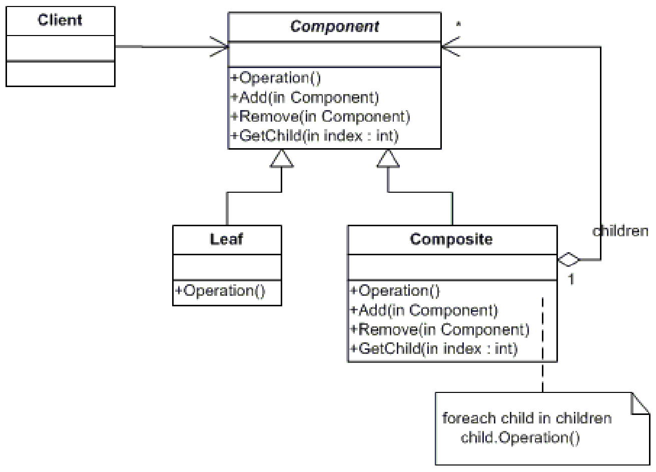
Object Adapter Pattern



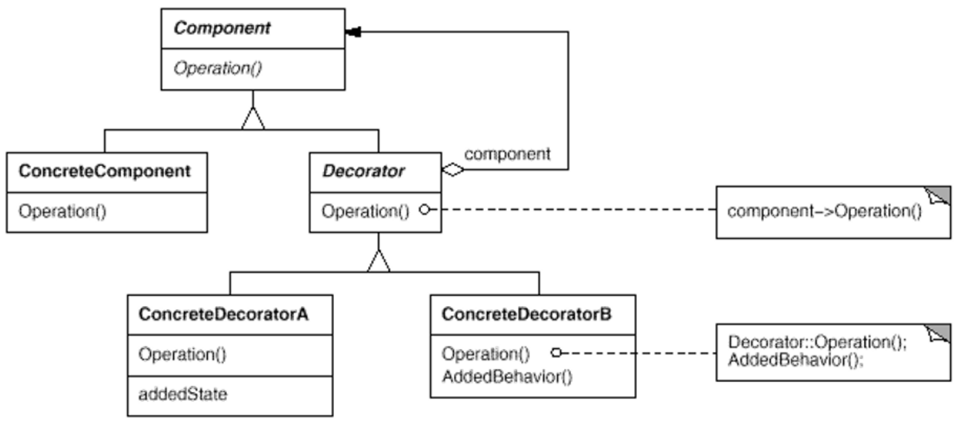
The Bridge Pattern



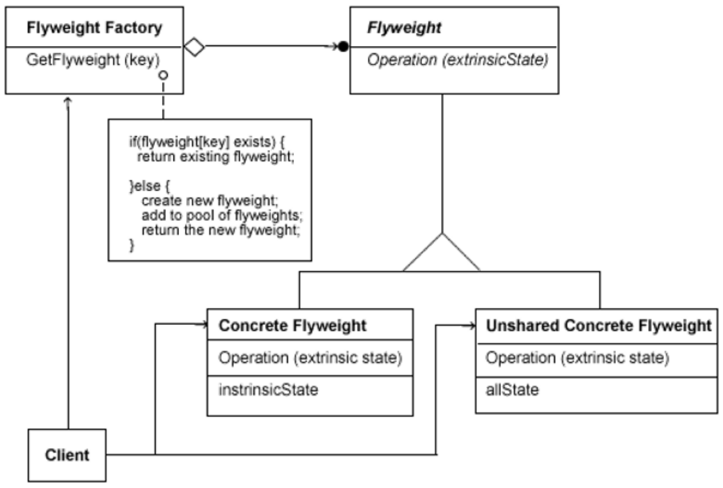
Composite:



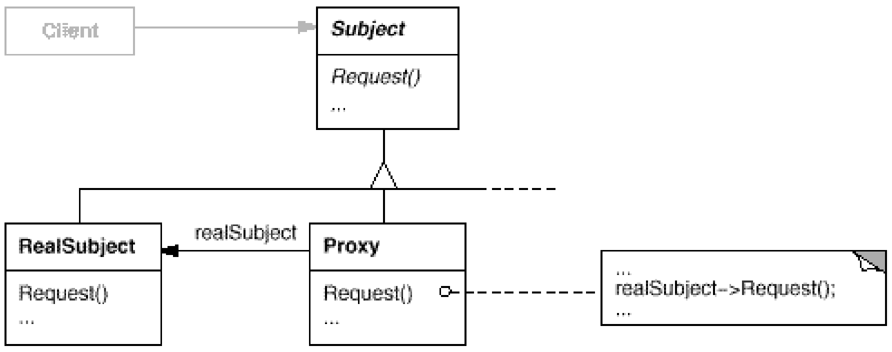
The Decorator Design Pattern



The Flyweight Pattern

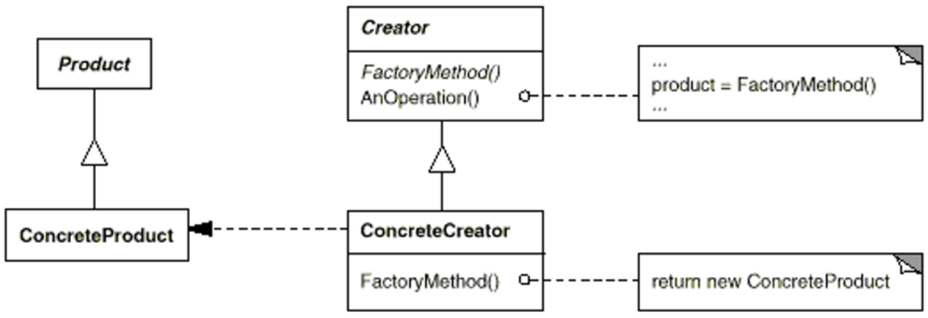

Flyweight Factory
GetFlyweight (key)
Flyweight
Operation (extrinsicState)
if(flyweight[key] exists) {
return existing flyweight;
}else {
create new flyweight;
add to pool of flyweights;
return the new flyweight;
}
Concrete Flyweight
Operation (extrinsic state)
instrinsicState
Unshared Concrete Flyweight
Operation (extrinsic state)
allState
Client

Proxy

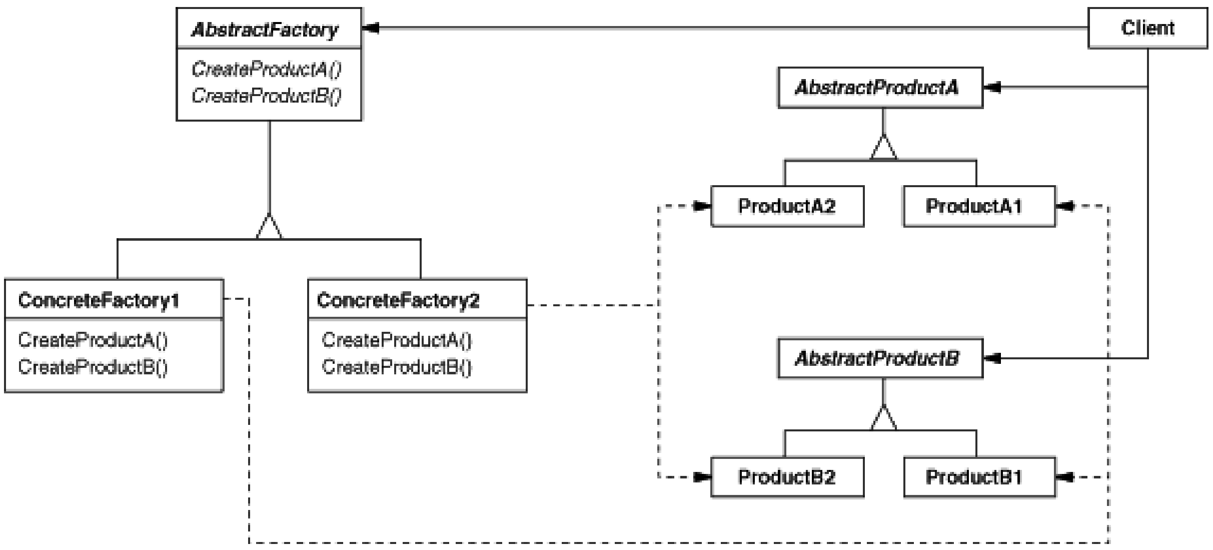


Factory Pattern – the solution

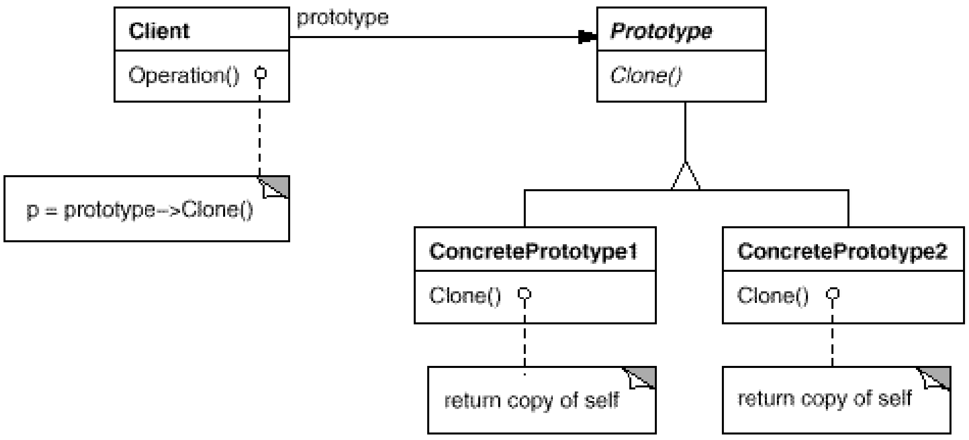
- Generally:



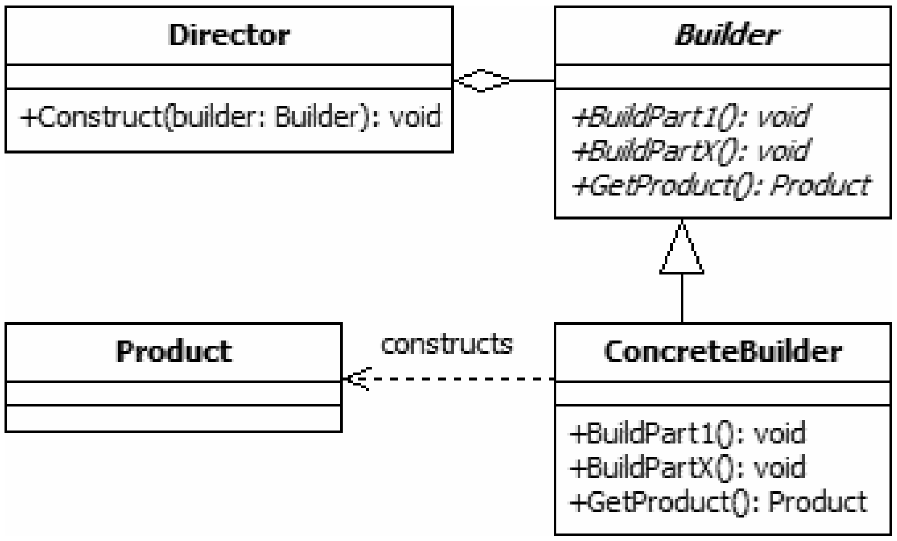
Abstract Factory



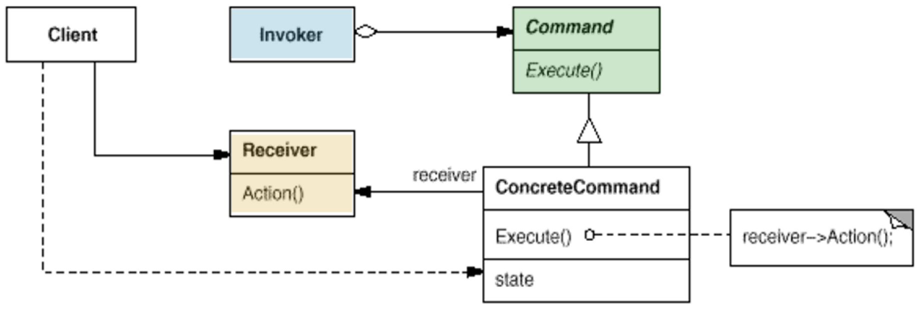
Prototype



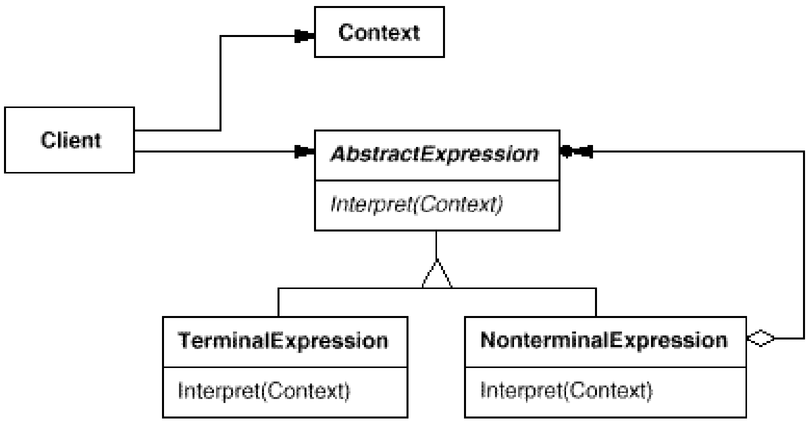
Builder



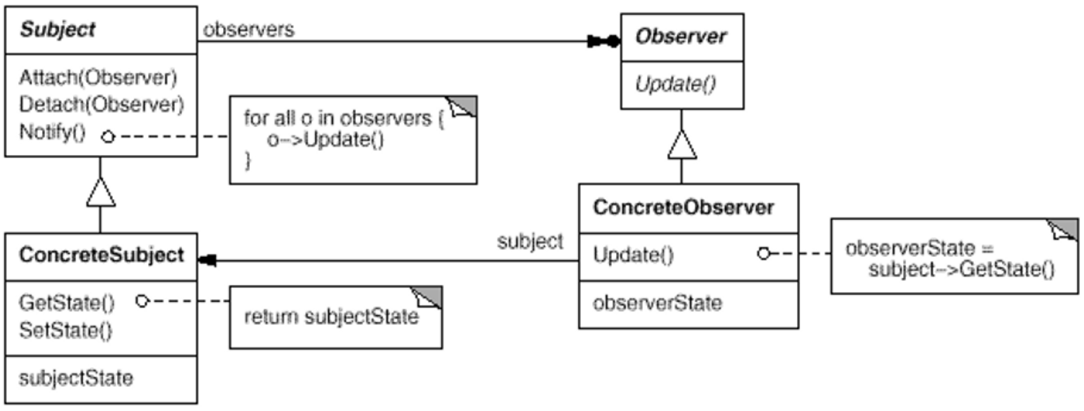
Command Pattern



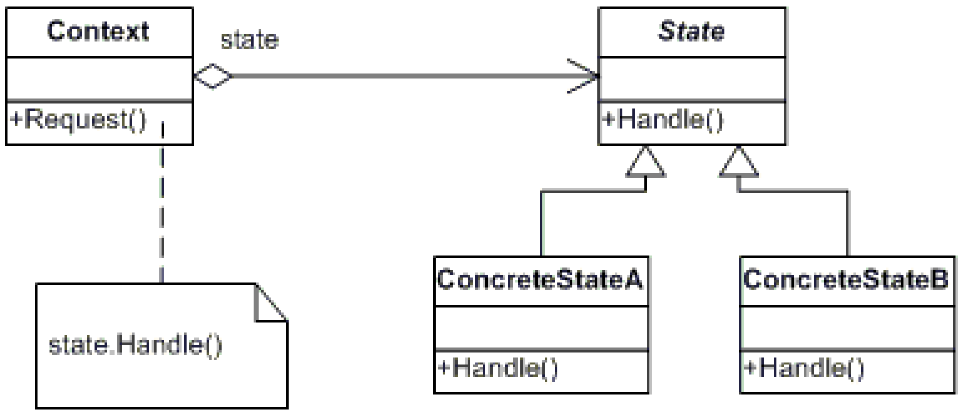
Interpreter:



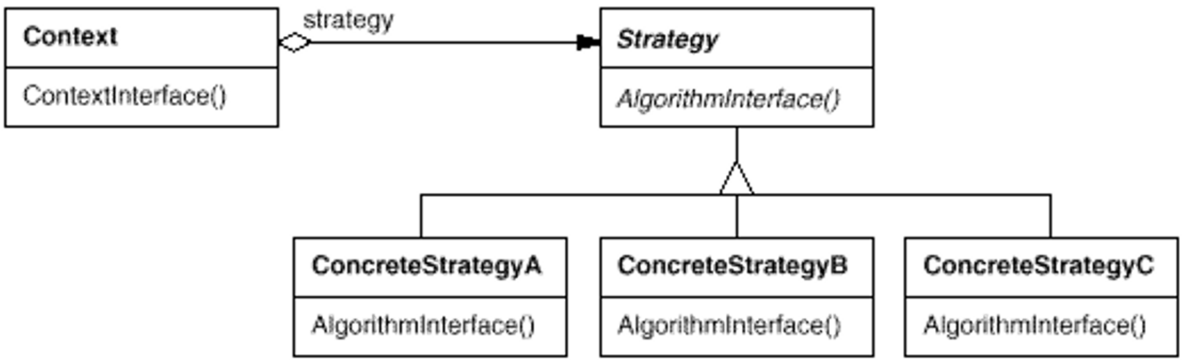
Observer Pattern



State pattern



Strategy:


Context
ContextInterface()
strategy
Strategy
AlgorithmInterface()
ConcreteStrategyA
AlgorithmInterface()
ConcreteStrategyB
AlgorithmInterface()
ConcreteStrategyC
AlgorithmInterface()