Creational Pattern

Builder Pattern

* Handles complex constructors
* Large number of parameters
* Immutability
* Examples:
  + StringBuilder
  + DocumentBuilder
  + Locale.Builder



Pros:

* Calls appropriate constructor
* Negates the need for exposed setters

Cons:

* Immutable
* Inner static class
* Designed first
* Complexity
* Method returns object

Example:

StringBuilder builder = new StringBuilder();

builder.append("This is an example ");

builder.append("of the builder pattern. ");

builder.append("It has methods to append ");

builder.append("almost anything we want to a String. ");

builder.append(42);

Prototype pattern

* Avoids costly creation
* Avoids subclassing
* Typically doesn’t use “new”
* Often utilizes an Interface
* Usually implemented with a Registry
* Example:
  + java.lang.Object#clone()



Pros:

* Clone / Cloneable
* Avoids keyword “new”
* Although a copy, each instance unique
* Costly construction not handled by client
* Can still utilize parameters for construction
* Shallow VS Deep Copy

Cons:

* Sometimes not clear when to use
* Used with other patterns
  + Registry
* Shallow VS Deep Copy

Example:

public class Statement implements Cloneable {

public Statement(String sql, List<String> parameters, Record record) {

this.sql = sql;

this.parameters = parameters;

this.record = record;

}

public Statement clone() {

try {

return (Statement) super.clone();

} catch (CloneNotSupportedException e) {}

return null;

}

}

Interpreter Pattern

* Represent grammar
* Interpret a sentence
* Map a domain
* AST
* Examples:
  + java.util.Pattern
  + java.text.Format



Pros:

* Access to properties
* Function as methods
* Adding new functionality changesevery variant

Examples:

String input = "Lions, and tigers, and bears! Oh, my!";

Pattern p =

Pattern.compile("(lion|cat|dog|wolf|bear|human|tiger|liger)");

Matcher m = p.matcher(input);

while (m.find()) {

System.out.println("Found a " + m.group() + ".");

}

Mediator Pattern

* Loose coupling
* Well-defined, but complex
* Reusable components
* Hub / Router
* Examples:
  + java.util.Timer
  + java.lang.reflect.Method#invoke()



Pros:

* Defines Interaction
* Object decoupling

Example:

public MediatorEverydayDemo(int seconds) {

toolkit = Toolkit.getDefaultToolkit();

timer = new Timer();

timer.schedule(new RemindTask(), seconds \* 1000);

timer.schedule(new RemindTaskWithoutBeep(), seconds \* 2 \* 1000);

}

class RemindTask extends TimerTask {

public void run() {

System.out.println("Time's up!");

toolkit.beep();

}

}