www.performance-lab.ru

PerformanceLab
Обеспечение производительности ИТ

Запуск внешних приложений

```
ProcessBuilder pb = new ProcessBuilder();
pb.command("ls" "-l")
     .directory(new File("/home")
     .redirectInput(Redirect.from(new File("cmd.txt"))
     .redirectOutput(Redirect.PIPE)
     .redirectError(Redirect.INHERIT);

Process process = pb.start();
```

Stream

Конструкция Stream представляет из себя вычисления в виде последовательности шагов (монада).
Stream является «одноразовым» и потенциально бесконечным.
Все операции делятся на промежуточные и терминальные.
Только одна терминальная операция может быть применена к одному инстансу.
Все промежуточные операции возвращают тот же самый объект, что позволяет группировать их в цепочки без использования точки с запятой.
Практически каждая коллекция может быть преобразована в Stream.
Есть отдельные виды стримов, к ним применяются специальные промежуточные операции (напр. IntPredicate) и поддерживают доп. терминальные операции: sum(), average().

Stream

Поставщики данных

```
Stream.of("qwe", "asd", "zxc"); //Стрим объектов
IntStream.range(8, 12) //Стрим целых чисел
    .mapToObj(i -> "c" + i); //Преобразуем в обычный стрим

//Используем поставщик данных
Stream<Integer> s = Stream.iterate(1, n -> n+1);
Stream<String> s = Stream.generate(() -> {return "abc";});
```

Singleton design pattern

Паттерн Singleton гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

```
class Singleton {
  private static Singleton instance;

  private Singleton() {};

  public static Singleton getInstance() {
    (if instance == null) instance = new Singleton;
     return instance;
  }

}
```

Fabric design pattern

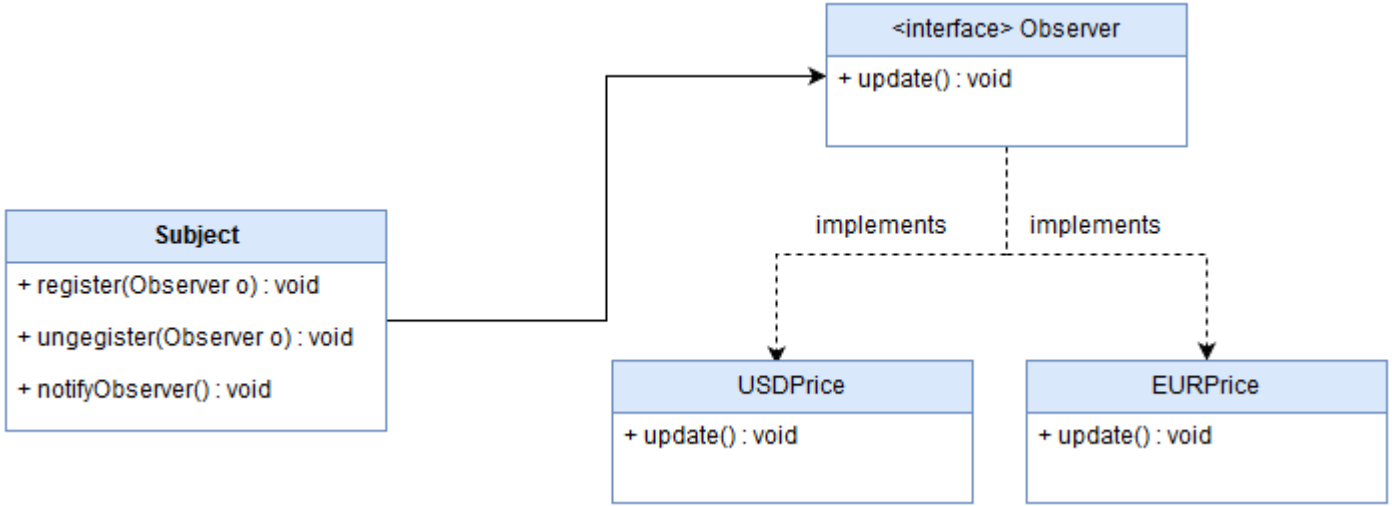
паттерн создания объектов (creational pattern). Данный шаблон проектирования предоставляет интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс инстанциировать.

Observer Design pattern

Применение

Используется когда есть несколько объектов, которые должны получить обновление, когда изменяется другой объект.

Субъект (издатель) ничего не знает о подписчиках (слабая связь)



Executor

Шаблон, предоставляющий абстракцию для работы с данными.

Удобно использовать при работе с данными (файлы, базы данных).

Executor

```
public interface ResultHandler<T> {
    T handle (ResultSet rs) throws SQLException;
}
public class Executor {
    public <T> T execQuery(
                                Connection cn,
                                String query,
                                ResultHandler<T> handler
    ) throws SQLException {
        Statement st = cn.createStatement();
        st.execute(query);
        ResultSet rs = st.getResultSet();
        T value = handler.handle(rs);
        rs.close(); st.close(); return value;
}}
```

Executor

```
Excecutor ex = new Executor();
String query = "SELECT user_name FROM users";
String userName = ex.execQuery(connection, query,
    new ResultHandler<String>(){
        @Override
        public String handle(ResultSet rs) {
            rs.next();
            return rs.getString("user_name");
        }
});
```

Reflection

Используется для получения информации о классе во время исполнения.

Получение объекта типа Class

```
Class peopleClass = People.class;
Class peopleClass = new People().getClass;
Class peopleClass = Class.forName("People");
```

Получение пакета

```
Package p = klass.getPackage();
```

Получение интерфейсов, реализумых классом:

```
Class[] interfaces = myClass.getInterfaces();
```

Reflection

Получение полей класса

```
Field[] fields = klass.getDeclaredFields();
```

Получение аннотаций метода

```
Annotation[] annotations = method.getAnnotations();
```

Получение конструкторов

```
Constructor[] cl = klass.getDeclaredConstructors();
```

Аннотации

Обычно используются для описания дополнительной конфигурации.
Могут иметь одно или несколько атрибутов.
Атрибуты ограничены примитивными типами, строками, типом Class<?>, перечислением (enum), аннотацией, массивом любого из этих элементов.
Для аннотаций могут применяться ограничения по применению или времени жизни.
Для значений может быть указано значение по-умолчанию. Если значение по-умолчанию не указано, оно становится обязательным, при использовании этой аннотации.

Аннотации

```
public @interface Version {
    String value() default "1.0.0";
}

public class Example {
    @Version("1.0.1")
    public firstMethod(){}
}
```

PerformanceLab

Обеспечение производительности ИТ

Спасибо за внимание!