System.arraycopy

System.arraycopy(source, startPos, dist, distPos, length);

- source - массив откуда нужно скопировать элементы начиная с позиции startPos и до позиции startPos + length.

- length - сколько элементов взять начиная от startPos.

- dist - массив, куда вставить скопированные элементы от source. Этот метод может работать с одним массивом для - source и dist.

- distPos - начиная с какого элемента вставлять скопированные ячейки.

|  |
| --- |
| package ru.job4j.tracker;  import java.util.Arrays;  public class DropArray {  public static void main(String[] args) {  String[] names = {"Petr", null, "Ivan", "Stepan", "Fedor"};  System.arraycopy(names, 2, names, 1, 3);  System.out.println(Arrays.toString(names));  }  } |

Scanner

Scanner input = new Scanner(System.in);

int select = Integer.valueOf(input.nextLine());

**Manifest**rseInt(input.nextLine());

Важной составляющей файла с расширением jar является наличие в нем файла MANIFEST.MF.

Манифест – это специальный файл, который содержит информацию о файлах, которые упакованы в jar-архиве. Это конфигурационный файл, который определяет поведение нашего jar-архива и определяет в себе ряд настроек.

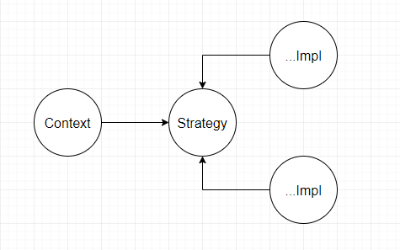
Очень важной настройкой, которую можно определить в файле MANIFEST.MF является точка входа в приложение, т.е. в нашем понимании это метод main(), который запускает все приложение. Если мы посмотрим на проект tracker, то это будет класс StartUI.

# шаблон – стратегия

Этот шаблон относится к поведенческим шаблонам.

Он позволяет изменить алгоритм работы программы в запущенной программе.

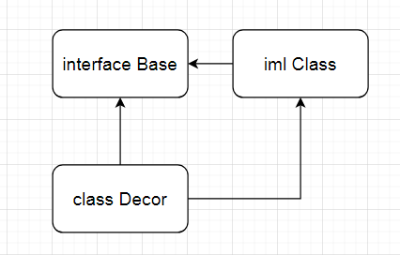
Жизненная аналогия. Телевизор показывает видео. Мы можем включить первый, второй и тд каналы и картинка всегда будет разная. Устройство воспроизведения одно, но картинки всегда разные.



# Шаблон Декоратор

Если рассматривать пример из жизни, это похоже на матрешку. Одна матрешка вкладывается в другую.  
Все выше описанное (смотри validateinput, в котором входящий параметр в конструкторе может быть consoleinput) есть шаблон проектирования "Декоратор".

Смысл Декоратора - добавление нового поведения в уже существующее поведение.



Base интерфейс описывающий поведение.

Class класс реализующий поведение.

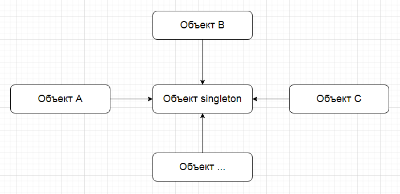
Decor класс реализующий поведение и зависимый от другой реализации Base.

# Шаблон singleton

используется для создания класса, который может содержать только один экземпляр объекта в виртуальной машине

Шаблон singleton применяют в проектах, где используются внешние ресурсы: файловая система, база данных, связь с другими приложениями.

В таких проектах доступ к ресурсу нужно ограничить использованием одного объекта на виртуальную машину.Это позволяет экономить ресурсы системы.



Пример:

|  |
| --- |
| ackage ru.job4j.tracker;  */\*\*  \* пример шаблона Singleton  \*/* public final class Log4File {   private static Log4File *instance* = null;   private String[] messages = new String[1000];  private int index = 0;   private Log4File() {   }   public static Log4File getInstance() {  if (*instance* == null) {  *instance* = new Log4File();  }  return *instance*;  }   public void add(String message) {  messages[index++] = message;  }   public void save() {  /\* *TODO Сохраняем записи из массива messages в файл.* \*/  } } |

# **Шаблон фабричный метод.**

Смысл этого шаблона в создании метода, который создает новый экземпляр объекта.

|  |
| --- |
| **public** **class** Shape { }  **public** **class** ShapeFactory {  **public** **static** Shape create(String name) {  Shape shape = **new** EmptyShape();  **if** (**"triangle"**.equals(name)) {  shape = **new** TriangleShape();  } **else** **if** (**"rectangle"**.equals(name)) {  shape = **new** RectangleShape();  }  **return** shape;  } }  **public** **static** **void** main(String[] args) {  Shape shape = ShapeFactory.create(**"cycle"**); } |

**Абстрактная фабрика.**

Главное отличие от фабричного метода, абстрактная фабрика это интерфейс, который содержит абстрактные методы для создания экземпляров.

|  |
| --- |
| **public** **interface** ShapeAbstFactory {  TriangleShape createTriangle();   RectangleShape createRectangle();   EmptyShape createEmpty(); }  **public** **class** ShapeFactory {  **private** **final** ShapeAbstFactory factory;   **public** ShapeFactory(ShapeAbstFactory factory) {  **this**.factory = factory;  }   **public** Shape create(String name) {  Shape shape = factory.createEmpty();  **if** (**"triangle"**.equals(name)) {  shape = factory.createTriangle();  } **else** **if** (**"rectangle"**.equals(name)) {  shape = factory.createRectangle();  }  **return** shape;  } } |

# StringBuilder

StringBuilder buffer = new StringBuilder();

buffer.append(obj.toString());

# Массив зеркало

|  |
| --- |
| public class BackArray {  public static void main(String[] args) {  String[] names = {"Petr", "Ivan", "Nik", "Vasya"};  int middle = names.length / 2;  for (int index = 0; index < middle; index++) {  String temp = names[index];  names[index] = names[names.length - 1 - index];  names[names.length - 1 - index] = temp;  }  System.out.println(Arrays.toString(names));  } } |

# Exception

встроенные классы, которые описывают исключительные ситуации(непроверяемые исключения):

**java.lang.IllegalArgumentException** **-** метод вызывается с некорректными параметрами**.**  
**java.lang.IllegalStateException** **-** метод вызывается с объекта в не корректном состоянии**.**  
**java.lang.NullPointerException** **-** методы вызывается у переменной, которая инициализирована **null** ссылкой.

Пример:

**throw** **new** IllegalArgumentException(**"Start should be less then finish."**);

# value.toCharArray();

Преобразует строку в массив символов char. Пример:

char[] check = value.toCharArray();

# Наследование, композиция и агрегация

эти термины описывают взаимодействия между объектами

**Наследование** - позволяет создавать новые объекты на основании ранее созданных.

**Композиция.** Другой способ сделать новый класс на основании другого класса.

(То есть дочерний объект содержит внутри родителя в качестве поля)

Как определить композицию? Если дочерний объект не может сущетсвовать без родителя, то такая связь называется композицией.

**Агрегация.** - Позволяет создавать объект, который содержит один или более объектов.

# String.format

String.format(**"Could not move by diagonal from %s to %s"**, source, dest)