

Паттерны проектирования: Singleton, Factory, FactoryMethod, AbstractFactory

Java Collections
7 уровень, 1 лекция

ОТКРЫТА

— Привет, Амиго!

— Привет, Билаабо!

— Сегодня у нас будет не просто интересная, а прямо-таки эпическая тема.

Сегодня я расскажу тебе, что такое **шаблоны проектирования (design patterns)**.

— Круто. Много про них слышал. Жду с нетерпением.

— Опытным программистам приходится писать очень много классов. Но самая сложная часть этой работы – это решать, какие классы должны быть и как распределить работу между ними.

Чем чаще они решали такие вопросы, тем чаще стали понимать, что существуют некоторые удачные решения, и наоборот, неудачные.

Неудачные решения обычно создают проблем больше, чем решают. Они плохо расширяются, создают много лишних ограничений, и т.п. А удачные решения – наоборот.

— А можно какую-нибудь аналогию?

— Допустим, ты строишь дом. И думаешь — из чего же он должен состоять. Ты решил, что тебе нужны стены, пол и потолок. В результате ты построил дом без фундамента и с ровной крышей. Такой дом будет трескаться, а крыша – протекать. Это – неудачное решение.

И наоборот: дом, состоящий из фундамента, стен и двускатной крыши будет удачным решением. Ему нестрашны ни большие снегопады (снег будет скатываться с крыши), ни подвижки почвы – фундамент будет обеспечивать стабильность. Такое решение мы назовем удачным.

— Ясно. Спасибо.

— Ок. Тогда я продолжу.

Со временем сборник удачных решений объявили «паттернами (шаблонами) проектирования», а сборник неудачных – антипаттернами.

Сам шаблон проектирования — это как бы ответ на вопрос. Его сложно понять, если не слышал самого вопроса.

Первая категория паттернов – это порождающие паттерны. Такие паттерны описывают удачные решения, связанные с созданием объектов.

— А что такого сложного в создании объектов?

— А вот как раз сейчас мы это и разберем.

Паттерн Singleton – Синглетон, Одиночка.



Часто в программе некоторые объекты могут существовать только в единственном экземпляре. Например, консоль, логгер, сборщик мусора и т.д.

Неудачное решение: отказаться от создания объектов, просто создать класс у которого все методы статические.

Удачное решение: создать единственный объект класса и хранить его в статической переменной. Запретить создание второго объекта этого класса. Пример:

Пример	
1	<code>class Sun</code>
2	<code>{</code>
3	<code>private static Sun instance;</code>
4	<code>public static Sun getInstance()</code>
5	<code>{</code>
6	<code>if (instance == null)</code>
7	<code>instance = new Sun();</code>
8	
9	<code>return instance;</code>
10	<code>}</code>
11	
12	<code>private Sun()</code>
13	<code>{</code>
14	<code>}</code>
15	<code>}</code>

Как вызвать	
1	<code>Sun sun = Sun.getInstance();</code>

Все просто.

Во-первых, мы сделали конструктор `private`. Теперь его можно вызвать только изнутри нашего класса. Мы запретили создание объекта `Sun` везде кроме методов класса `Sun`.

Во-вторых, получить объект этого класса можно только вызвав метод `getInstance()`. Это не только единственный метод

НАЧАТЬ ОБУЧЕНИЕ

— Ясно.

— Когда человек думает – как же именно это сделать? Паттерн говорит – можешь попробовать так – это одно из удачных решений.

— Спасибо. Теперь что-то начинает проясняться.

— Также про этот паттерн можно прочитать [здесь](#).

Паттерн Factory – Фэктори, Фабрика.



Очень часто программисты сталкиваются вот с какой ситуацией. У тебя есть некоторый базовый класс и много подклассов. Например – персонаж игры – GamePerson и классы всех остальных персонажей игры, унаследованные от него.

Допустим, у тебя есть такие классы:

Пример	
1	abstract class GamePerson
2	{
3	}
4	
5	class Warrior extends GamePerson
6	{
7	}
8	
9	class Mag extends GamePerson
10	{
11	}
12	
13	class Troll extends GamePerson
14	{
15	}
16	
17	class Elv extends GamePerson
18	{
19	}

НАЧАТЬ ОБУЧЕНИЕ

Если проблема кажется тебе надуманной, представь, что в игре нужно создавать десятки мечей и щитов, сотни магических заклинаний, тысячи монстров. Без удобного подхода к созданию объектов тут не обойтись.

Вот какое «удачное решение» предлагает паттерн **Фабрика (Factory)**.

Во-первых, надо завести enum, значения которого будут соответствовать различным классам.

Во-вторых, сделать специальный класс – **Factory**, у которого будет статический метод или методы, которые и будут заниматься созданием объекта(ов) в зависимости от enum’а.

Пример:

Пример	
1	public enum PersonType
2	{
3	UNKNOWN,
4	WARRIOR,
5	MAG,
6	TROLL,
7	ELV,
8	}
9	
10	public class PersonFactory
11	{
12	public static GamePerson createPerson (PersonType personType)
13	{
14	switch (personType)
15	{
16	WARRIOR:
17	return new Warrior();
18	MAG:
19	return new Mag();
20	TROLL:
21	return new Troll();
22	ELV:
23	return new Elv();
24	default:
25	throw new GameException();
26	}
27	}
28	}

Как вызывать	
1	GamePerson person = PersonFactory. createPerson (PersonType.MAG);

— Т.е. мы создали специальный класс для управления созданием объектов?

— Ага.

— А какие преимущества это дает?

— Во-первых, там эти объекты можно инициализировать нужными данными.

Во-вторых, между методами можно сколько угодно передавать нужный enum, чтобы в конце концов по нему создали правильный объект.

В третьих, количество значений enum не обязательно должно совпадать с количеством классов. Типов персонажей может быть много, а классов – мало.

Например, для **Mag & Warrior** можно использовать один класс – Human, но с различными настройками силы и магии (параметрами конструктора).

Вот как это может выглядеть (для наглядности добавил еще темных эльфов):

Пример	
1	public enum PersonType
2	{
3	UNKNOWN,
4	WARRIOR,
5	MAG,
6	TROLL,
7	ELV,
8	DARK_ELV
9	}
10	
11	public class PersonFactory
12	{
13	public static GamePerson createPerson (PersonType personType)
14	{
15	switch (personType)
16	{
17	WARRIOR:
18	return new Human(10, 0); //сила, магия
19	MAG:
20	return new Human(0, 10); //сила, магия
21	TROLL:
22	OGR:
23	return new Troll();
24	ELV:
25	return new Elv(true); //true – добрый, false – злой
26	DARK_ELV:
27	return new Elv(false); //true – добрый, false – злой
28	default:
29	throw new GameException();
30	}
31	}
32	}

Как вызывать	
1	GamePerson person = PersonFactory.createPerson(PersonType.MAG);

В примере выше мы использовали всего три класса, чтобы создавать объекты шести разных типов. Это очень удобно. Более того, у нас вся эта логика сосредоточена в одном классе и в одном методе.

Если мы вдруг решим создать отдельный класс для Огра, мы просто поменяем тут пару строк кода, а не будем перелопачивать половину приложения.

— [Согласен. Удачное решение.](#)

— А я тебе о чем говорю, шаблоны проектирования – это сборники удачных решений.

НАЧАТЬ ОБУЧЕНИЕ

— Ага. Сходу не разберёшься, согласен. Но все равно, лучше знать и не уметь, чем не знать и не уметь. Вот тебе еще полезная ссылка по этому паттерну: [Паттерн Factory](#).

— О, спасибо.

— Паттерн **Abstract Factory – Абстрактная Фабрика**.

Иногда, когда объектов очень много, напрашивается создание фабрики фабрик. Такую фабрику принято называть **Абстрактной Фабрикой**.

— Это ж где такое может понадобиться?

— Ну, например, у тебя есть несколько групп идентичных объектов. Это легче показать на примере.

Смотри, допустим, у тебя в игре есть три расы – люди, эльфы и демоны. И у каждой расы для баланса есть воин, лучник и маг. В зависимости от того, за какую сторону играет человек, он может создавать только объекты своей расы. Вот как это могло бы выглядеть в коде:

Объявление классов войск	
1	class Warrior
2	{
3	}
4	class Archer
5	{
6	}
7	class Mag
8	{
9	}

Люди	
1	class HumanWarrior extends Warrior
2	{
3	}
4	
5	class HumanArcher extends Archer
6	{
7	}
8	
9	class HumanMag extends Mag
10	{
11	}

Эльфы	
1	class ElvWarrior extends Warrior
2	{
3	}
4	
5	class ElvArcher extends Archer
6	{
7	}
8	
9	class ElvMag extends Mag
10	{
11	}

Демоны

1

class DaemonWarrior extends Warrior

2

{

3

}

4

5

class DaemonArcher extends Archer

6

{

7

}

8

9

class DaemonMag extends Mag

10

{

11

}

А теперь создадим расы, ну или можно еще назвать их армиями.

Армии

1

abstract class Army

2

{

3

public Warrior createWarrior();

4

public Archer createArcher();

5

public Mag createMag();

6

}

Армия людей

1

class HumanArmy extends Army

2

{

3

public Warrior createWarrior()

4

{

5

return new HumanWarrior();

6

}

7

public Archer createArcher()

8

{

9

return new HumanArcher();

10

}

11

public Mag createMag()

12

{

13

return new HumanMag();

14

}

15

}

Армия эльфов

1

class ElvArmy extends Army

2

{

3

public Warrior createWarrior()

4

{

5

return new ElvWarrior();

6

}

7

public Archer createArcher()

8

{

9

return new ElvArcher();

12	{
13	return new ElvMag();
14	}
15	}

Армия демонов	
1	class DaemonArmy extends Army
2	{
3	public Warrior createWarrior()
4	{
5	return new DaemonWarrior();
6	}
7	public Archer createArcher()
8	{
9	return new DaemonArcher();
10	}
11	public Mag createMag()
12	{
13	return new DaemonMag();
14	}
15	}

— А как это использовать?

— Можно везде в программе использовать классы Army, Warrior, Archer, Mag, а для создания нужных объектов – просто передавать объект нужного класса-наследника Army.

Пример:

Пример	
1	Army humans = new HumanArmy();
2	Army daemons = new DaemonArmy();
3	
4	Army winner = FightSimulator.simulate(humans, daemons);

В примере выше у нас есть класс, который симулирует бои между разными расами (армиями), и ему нужно просто передать два объекта Army. С их помощью он сам создает различные войска и проводит виртуальные бои между ними, с целью выявить победителя.

— Ясно. Спасибо. Действительно интересный подход.

Удачное решение, что ни говори.

— Ага.

Вот еще хорошая ссылка по этой теме: [Паттерн Abstract Factory](#)

 +53 

Комментарии (34)

популярные новые старые

JavaCoder

Введите текст комментария

Andrey Karelin Уровень 41, Сумы, Украина 6 мая, 19:27 

Тема "Абстрактной фабрики" не раскрыта

Ответить  +5 



LuneFox инженер по сопровождению в BIFIT EXPERT 24 февраля, 17:07 

MAG, ELV, OGR)))000

Ответить  +2 

Ars Уровень 41 26 ноября 2021, 16:16 

Как всегда, сначала задачи и самостоятельное изучение. Потом лекция и преподнесение материала как нового.

Ответить  +6 

Vladrip Уровень 41, Ивано-Франковск, Украина 11 октября 2021, 20:19 

Вместо demon даемон, это многопоточность так влияет?))))

Ответить  0 

Anton Stezhkin Уровень 41 31 июля 2021, 17:11 

Если вдруг еще кто-то не читал, рекомендую [Refactoring-guru](#)

Ответить  +3 

ram0973 Уровень 41, Набережные Челны, Россия 14 августа 2020, 14:16 

[Шпаргалка](#) Можно распечатать

Ответить  +32 

Anna Avilova architect 26 апреля, 16:00 

большое человеческое спасибо!

Ответить  0 

Oleg Khilko Уровень 51, Москва, Russian Federation 13 августа, 11:08 

А помнишь этот момент? [Американский психопат — сравнения визитных карточек](#)
[Шпаргалка на русском](#) можно распечатать

Ответить  0 

Артур Харатян Уровень 41, Санкт-Петербург, Россия 12 мая 2020, 09:48 

Почему Билабо хочется прочитать балабол?

Ответить  +5 

Regina Support в JavaRush MODERATOR 13 мая 2020, 13:38 

Это Вы еще Хулио не читали.)))

[Отличный сайт с разными шаблонами с примером кода](#)

Ответить

+9

Валентин Кудинов

Уровень 41, Самара, Россия

31 марта 2020, 12:22

Ссылки не открываются.
Разработчики JavaRush прошу исправить.

Ответить

+4

Roman

Уровень 32, Германия

11 марта 2020, 18:45

Уважаемые авторы, вычитывайте, пожалуйста, тексты лекций внимательнее!
Mag - Magician (а не Mag)
Эльф - Elf (не Elv)
Singleton - Синглтон (не синглЕтон)

А еще все ссылки возвращают 502 :(Может вам имеет смысл повесить простенький мониторинг на сайты из курса? (uptimerobot какой-нибудь)

Ответить

+4

Эли Гутман

Уровень 37

28 марта 2021, 18:34

Magician - в первую очередь скорее фокусник, а реальный фэнтезийный маг - mage. Можно погуглить, например, картинки по обоим словам.

Ответить

0

Показать еще комментарии

- ОБУЧЕНИЕ
- Курсы программирования
- Курс Java
- Помощь по задачам
- Подписки
- Задачи-игры

- СООБЩЕСТВО
- Пользователи
- Статьи
- Форум
- Чат
- Истории успеха
- Активности

- КОМПАНИЯ
- О нас
- Контакты
- Отзывы
- FAQ
- Поддержка



JavaRush — это интерактивный онлайн-курс по изучению Java-программирования с нуля. Он содержит 1200 практических задач с проверкой решения в один клик, необходимый минимум теории по основам Java и мотивирующие фишки, которые помогут пройти курс до конца: игры, опросы, интересные проекты и статьи об эффективном обучении и карьере Java-девелопера.

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский

СКАЧИВАЙТЕ НАШИ ПРИЛОЖЕНИЯ



НАЧАТЬ ОБУЧЕНИЕ

