НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Объектно-ориентированное программирование на языке Java

Часть 2. Введение в объекты

Yevhen Berkunskyi, NUoS
eugeny.berkunsky@gmail.com
http://www.berkut.mk.ua

Развитие абстракции

- Язык Ассемблер – является небольшой абстракцией базовой машины, на котором он работает.
- Многие так называемые «командные» языки, которые были созданы вслед за ним (например, FORTRAN, BASIC и C), представляли абстракцию следующего уровня.
- Объектно-ориентированный подход идет еще дальше, предоставляя программисту средства для представления задачи в ее пространстве.

Развитие абстракции

- ООП позволяет описать проблему с точки зрения проблемы, а не с точки зрения компьютера, на котором будет исполнено решение.
- Впрочем, связь с компьютером сохранилась:

Каждый объект похож на маленький компьютер - у него есть состояние, и у него есть операции, которые он может выполнить

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

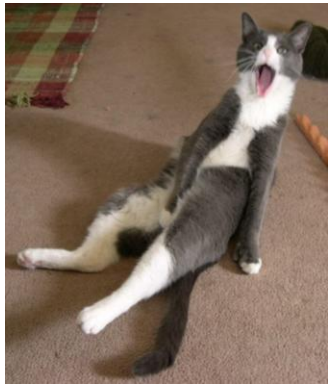
Характеристики ООП

1. Все является объектом
2. Программа –это группа объектов, указывающих друг другу, что делать, посредством сообщений
3. Каждый имеет свою собственную память, состоящую из других объектов
4. Каждый объект имеет свой тип
5. Все объекты определенного типа могут получать одни и те же сообщения

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Что такое объект?

Каждый объект обладает состоянием, поведением и индивидуальностью



Это означает, что у объекта могут быть внутренние данные (которые дают ему состояние), методы (для создания поведения), и каждый объект можно отличить от любого другого объекта (каждый объект имеет свой уникальный адрес в памяти)

У объектов есть интерфейс

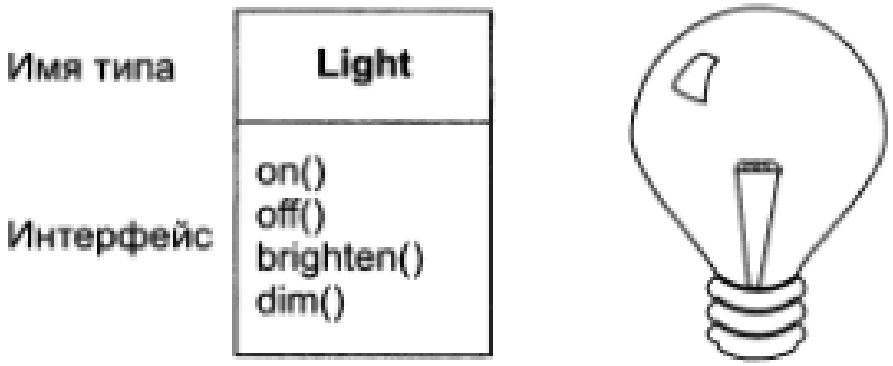
В объектно-ориентированном программировании мы создаем новые типы данных, во всех объектно-ориентированных языках программирования используется ключевое слово «class».

Когда вы видите слово «тип», думайте «класс» и наоборот

После определения класса вы можете создать произвольное количество объектов этого класса, а затем манипулировать этими объектами, как если бы они представляли собой элементы решаемой задачи.

У объектов есть интерфейс

Каждый объект умеет выполнять только определенный круг запросов. Запросы, которые вы можете посылать объекту, определяются его *интерфейсом*, причем интерфейс объекта определяется типом.




```
Light lt = new Light();
lt.on();
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Повторное использование

Композиция (Агрегирование)

“has-a” («содержит что-то»)

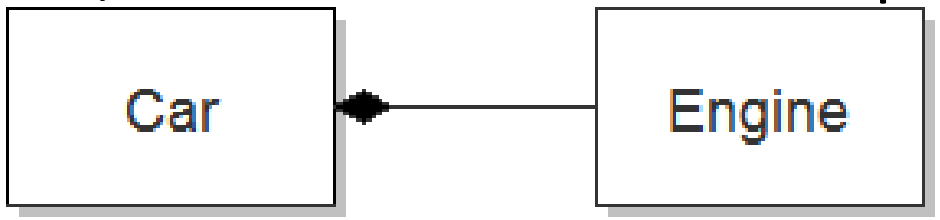
Наследование

“is-a” («является чем-то»)

Композиция (Агрегирование)

Самый простой способ использовать класс повторно, это создать несколько объектов данного типа, но можно поместить объект внутрь нового класса.

Поскольку вы составляете новый класс из уже существующих классов, это понятие называется композицией (если композиция происходит динамически, ее обычно называют агрегацией).



Композиция часто упоминается как отношение «has-a», как в предложении «Автомобиль имеет двигатель»,

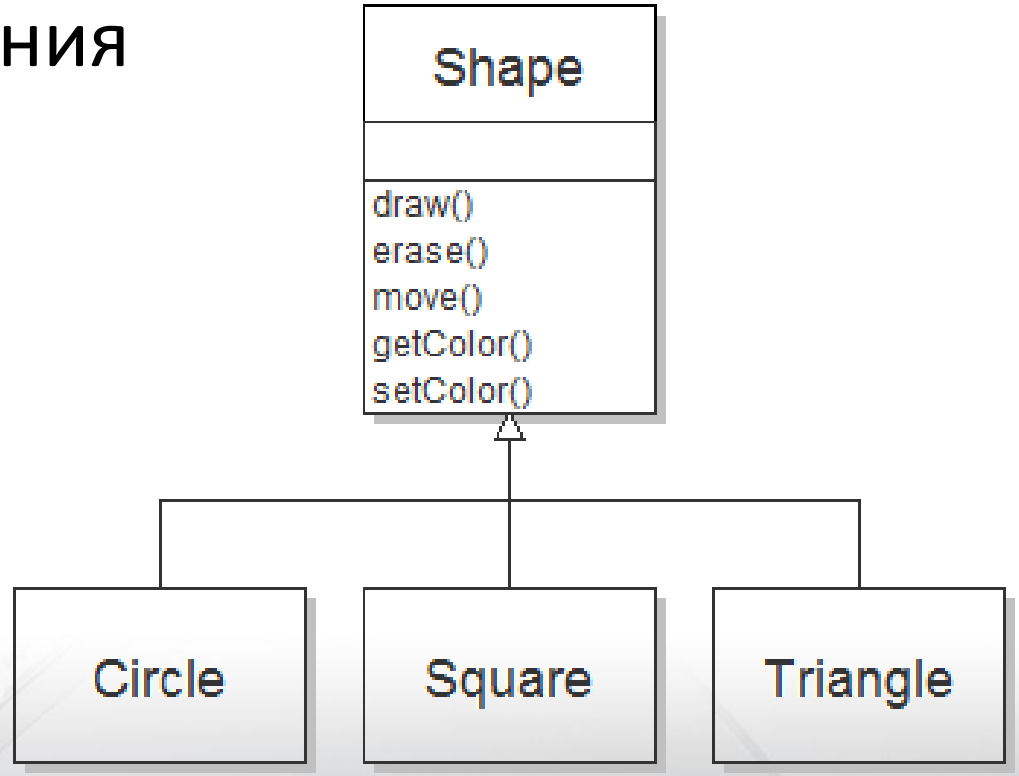
Композиция (Агрегирование)

```
public class Engine {
    int type;
    String series;
    double power;
}

public class Car {
Engine engine;
....
}
```

Наследование

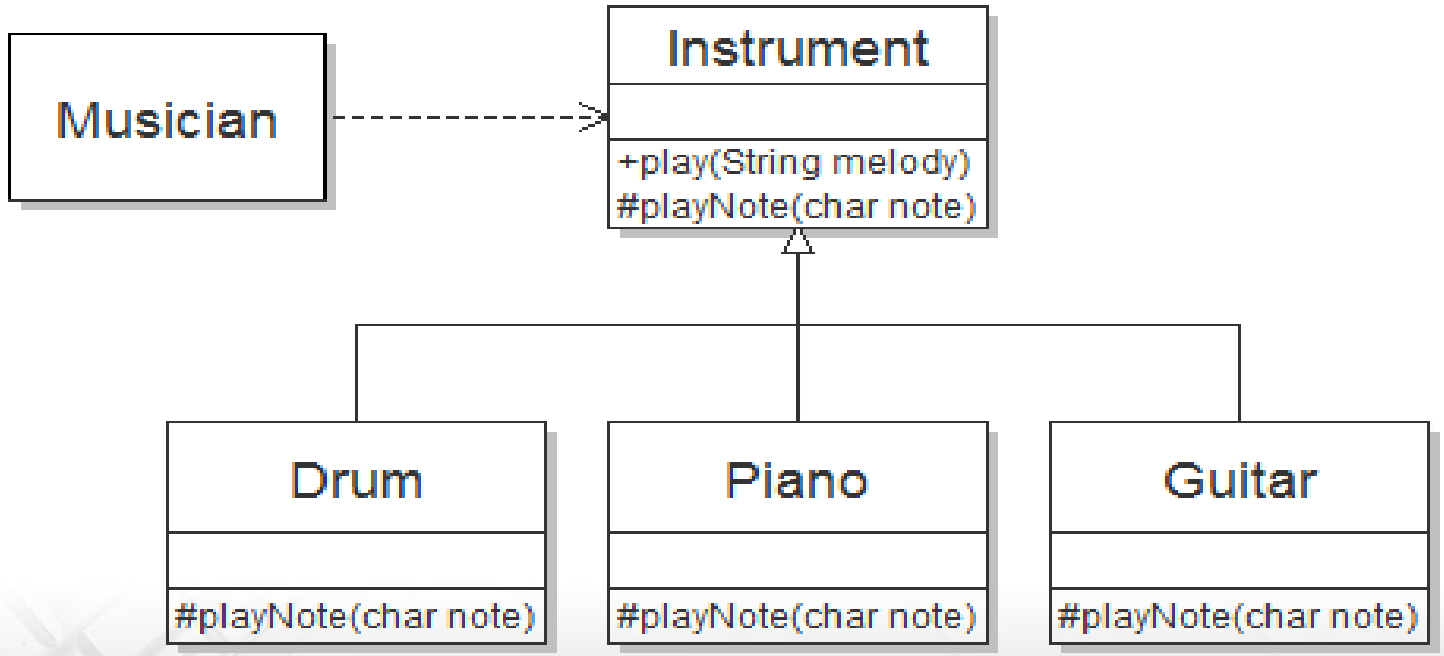
Мы можем взять существующий класс, «клонировать» его, а затем внести дополнения и обновления



НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Полиморфизм

Давайте рассмотрим музыканта, который пользуется музыкальными инструментами (он может нажимать на клавиши, бить по струнам, нам не нужно придумывать массу методов)



Основы Java

Хотя вы обращаетесь со всем как с объектом, на самом деле он представляет собой *ссылку* на объект

Все объекты должны создаваться явно

Когда вы создаете ссылку, вам необходимо связать ее с новым объектом. В основном это делается с помощью оператора **new** :

```
 String s = new String("asdf");
var s =new String("asdf");// JAVA10
 Scanner in = new Scanner(System.in);
```

Основы Java

Особый случай: примитивные типы

В Java размеры каждого примитивного типа строго фиксированы.

Они не меняются при переходе на иную машинную архитектуру, как в большинстве языков.

Незыблемость размера – одна из причин улучшенной переносимости Java-программ .

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Примитивные типы

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	—	—	—	**Boolean**
char	16 bits	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8 bits	-128	+127	**Byte**
short	16 bits	-2^{15}	$+2^{15}$-1	**Short**
int	32 bits	-2^{31}	$+2^{31}$-1	**Integer**
long	64 bits	-2^{63}	$+2^{63}$-1	**Long**
float	32 bits	IEEE754	IEEE754	**Float**
double	64 bits	IEEE754	IEEE754	**Double**
void	—	—	—	**Void**

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Массивы в Java

- Массив в **java** гарантированно инициализируется, к нему невозможен доступ за пределами его границ.
- Проверка границ массива обходится относительно дорого, как и проверка индекса во время выполнения, но предполагается, что повышение безопасности и подъем производительности стоят того.

Массивы в Java

- Все массивы (как массивы примитивов, так и массивы объектов) содержат поле, которое можно прочитать (но не изменить!) для получения количества элементов в массиве.

Это поле называется **length. Java** защищает вас от проблем — при выходе за рамки массива происходит ошибка времени исполнения (*исключение*)

Массивы в Java

- Когда вы создаете массив объектов, вы на самом деле создаете массив ссылок, и каждая из этих ссылок автоматически инициализируется специальным значением, представленным ключевым словом: null
- Вы также можете создать массив примитивов. И снова, компилятор гарантирует инициализацию, поскольку память для этого массива заполнится нулями.

```
Instrument[] ensemble = new Instrument[5];
int[] nums = new int[10];
double[] x = {0.1, -0.4, 0.6, 0.2};
```

Модификаторы доступа

- В java существуют 4 модификатора доступа **private, default, protected и public**. Их можно применять как к полям класса так и к методам, а также к самим классам и интерфейсам.

Далее будем говорить только о членах класса – полях и методах.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Модификаторы доступа

	В том же классе	Из другого класса		Из потомка этого класса (через наследование)	
		В том же пакете	В другом пакете	В том же пакете	В другом пакете
private	Доступ есть	Доступа нет	Доступа нет	Доступа нет	Доступа нет
default	Доступ есть	Доступ есть	Доступа нет	Доступ есть	Доступа нет
protected	Доступ есть	Доступ есть	Доступа нет	Доступ есть	Доступ есть
public	Доступ есть	Доступ есть	Доступ есть	Доступ есть	Доступ есть

- В таблице сведено применение различных модификаторов к классам, полям, методам, конструкторам и блокам.
- Уровень доступа **private** используется для сокрытия методов или полей класса от других классов, т.е. они доступны только внутри класса. Если необходимо получить доступ к этим полям, то определяют внутри класса специальные методы: *getter* и *setter*.
- Модификатор **protected** используется для определения видимости членов класса в самом классе и в его наследниках.
- Модификатор доступа **public** означает, что метод или поле видны и доступны любому классу. В одном файле может быть только один public класс