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

- 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

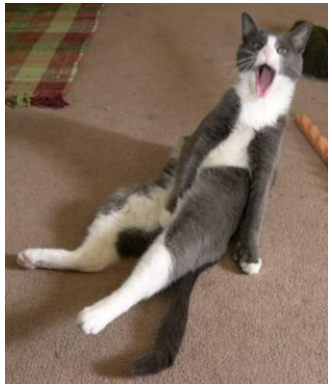
Характеристики ООП

1. Все является объектом
2. Программа –это группа объектов, указывающих друг другу, что делать, посредством сообщений
3. Каждый объект 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)

Что такое объект?

- *Поведение* объекта — что с ним можно делать и какие методы к нему можно применять.
- *Состояние* объекта — как этот объект реагирует на применение методов.
- *Идентичность* объекта — чем данный объект отличается от других, характеризующихся таким же поведением и состоянием.

Что такое объект?

Все объекты, являющиеся экземплярами одного и того же класса, ведут себя одинаково. *Поведение* объекта определяется методами, которые можно вызвать. Каждый объект сохраняет информацию о своем *состоянии.*

Со временем состояние объекта может измениться, но спонтанно это произойти не может. Состояние объекта может изменяться только в результате вызовов методов. (Если состояние объекта изменилось вследствие иных причин, значит, принцип инкапсуляции не соблюден.)

Что такое объект?

Состояние объекта не полностью описывает его, поскольку каждый объект имеет свою собственную *идентичность.*

Например, в системе обработки заказов два заказа могут отличаться друг от друга, даже если они относятся к одним и тем же товарам.

Заметим, что индивидуальные объекты, представляющие собой экземпляры класса, *всегда* отличаются своей идентичностью и, как правило, — своим состоянием.

Основы Java

Ссылки на объекты

- В Java для манипулирования объектами в программном коде используются *ссылки на объекты (handles)*. Ссылка хранит в себе некоторый адрес объекта в оперативной памяти.
- Может быть несколько ссылок на один объект. На какой-то объект может вообще не быть ссылок (тогда он для нас безвозвратно потерян). Ссылка может не ссылаться ни на какой объект — пустая (**null**) ссылка.

Основы Java

Все ссылки имеют имя

- Все ссылки, так или иначе, описываются, при этом каждой ссылке дается имя.

Все ссылки строго типизированы

- При описании ссылки обязательно указывается ее тип. И эта ссылка может ссылаться только на объект данного типа. Есть определенные исключения из этого правила, связанные с наследованием классов. Попытка присвоить ссылке адрес объекта не того класса возникает исключительная ситуация ClassCastException

Основы Java

Приведем пример описания ссылки

```
Main prog;
```

Здесь `Main` — имя типа (как и ссылки, все типы имеют имя), `prog`— имя ссылки. После такого описания ссылке `prog` можно присвоить значение — адрес какого-то объекта типа `Main`.

Основы Java

Создание объектов

Все объекты в Java создаются только явно, для чего используется операция **`new`**.

```
prog = new Main();
```

Здесь создается объект типа `Main` и его адрес заносится в `prog`.

Еще один пример

```
Main prog = new Main();
```

Здесь описание ссылки совмещено с инициализацией.

Основы Java

Класс — способ описания типа

Для описания типов в Java используется механизм **классов**. За исключением базовых (иначе — элементарных) типов (int, char, float и др.), все остальные типы — это классы.

В простейшем случае описание класса выглядит так

```
class MyClass {. . . // тело класса }
```

Здесь **class** — ключевое слово, MyClass — имя класса. Внутри фигурных скобок находится тело класса.

Внутри тела класса описываются в произвольном порядке поля и методы класса.

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

Рассмотрим сначала простейшие одномерные массивы базовых типов.

```
int intAry[]; или
int[] intAry;
```

Это описание ссылки на массив `intAry`. Соответственно, в этом описании размер массива не указывается и сам массив не создается. Как и любой другой объект массив должен быть создан операцией **new** .

```
intAry = new int[10];
```

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};
```

Массивы объектов

Одномерный массив объектов — это массив ссылок на объекты. Соответственно, **нужно создать как массив, так и сами объекты.**

Вариант 1. (явное создание)

```
SomeClass objAry[] = new SomeClass[4];
for (int j = 0; j < objAry.length; j++ )
   objAry[j] = new SomeClass();
```

Массивы объектов

Вариант 2. (использование списка инициализации)

```
SomeClass objAry[] = new SomeClass[]
{
     new SomeClass(),
     new SomeClass(),
     new SomeClass(),
     new SomeClass(),
     };
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Многомерные массивы

Массив является объектом.
Двумерный массив — это массив ссылок на объекты-массивы.
Трехмерный массив — это массив ссылок на массивы, которые, в свою очередь, являются массивами ссылок на массивы.

Вариант 1. (явное создание)

```
int ary[][] = new int[3][3];
```

Внимание: в варианте 1 массив создан, но его элементы имеют неопределенное значение. Если попытаться их использовать, возникнет Exception.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Многомерные массивы

Вариант 2. (использование списка инициализации)

```
int ary1[][] = new int[][] {
                    {1, 1, 1},
                    {2, 2, 2},
                    {1, 2, 3}
                    };
int ary2[][] = new int[][] {
                   {1, 1, 1, 1},
                   {2, 2, 2},
                   {1, 2, 3, 4, 5}
                 };
```

Многомерные массивы

Создание таких "непрямоугольных" массивов возможно не только с использованием списка инициализации, но и явно.

```
int ary[][] = new int[3][];
    ary[0] = new int[5];
    ary[1] = new int[2];
    ary[2] = new int[6];
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Присваивание и копирование массивов

В приведенных выше примерах имя массива — это переменная-ссылка (или поле-ссылка), содержащая адрес объекта массива. Поэтому присваивание таких переменных друг другу — это не копирование массивов, а копирование ссылок на объекты. Например.

```
int ary[][] = new int[][] {
                    {1, 1, 1},
                    {2, 2, 2},
                    {1, 2, 5}};
int copyAry[][] = ary;
```

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

класс Arrays

`copyAry` — это не ссылка на копию массива, а еще одна ссылка на тот же массив. Для создания копии придется написать соответствующий метод.

В состав стандартной библиотеки Java входят разнообразные средства работы с массивами. В пакете java.util имеется класс Arrays, который обеспечивает множество полезных операций над массивами

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

java.util.Arrays

[]b=Arrays.copyOf([]a, int newLength)	копирование массива, []a – исходный массив []b – новый массив newLength – длина нового массива
[]b= a.java.lang.Object.clone()	копирование массива []a – исходный массив []b – новый массив (new не нужно использовать)
Arrays.sort([]a)	Сортировка. Упорядочивание всего массива в порядке возрастания
Arrays.sort([]a,index1,index2)	Сортировка части массива в порядке возрастания
Arrays.sort([]a, Collections.reverseOrder());	Сортировка. Упорядочивание всего массива в порядке убывания

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

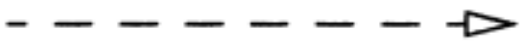
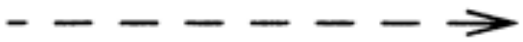
java.util.Arrays

Boolean f=Arrays.equals([]a,[]b)	сравнение массивов
String str=Arrays.toString([]a);	Вывод одномерных массивов. Все элементы представлены в виде одной строки
int index=Arrays.binarySearch([]a,элемент a)	поиск элемента методом бинарного поиска
Arrays.fill([]a, элемент заполнения)	заполнение массива переданным значением
Boolean f=Arrays.deepEquals([]a, []b)	сравнение двумерных массивов
deepToString([] a)	Преобразовывает 2-мерный массив в строку

Отношения между классами

Между классами существуют три общих вида отношений.

- Зависимость ("использует — что-то").
- Агрегирование ("содержит — что-то").
- Наследование ("является — чем-то").

Отношение	Обозначение в UML
Наследование	
Реализация интерфейса	
Зависимость	
Агрегирование	
Связь	
Направленная связь	

У объектов есть интерфейс

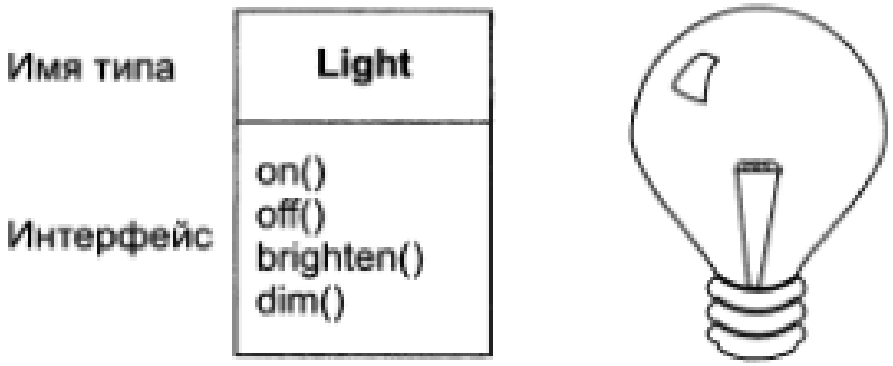
В объектно-ориентированном программировании мы создаем новые типы данных, во всех объектно-ориентированных языках программирования используется ключевое слово «class».

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»)

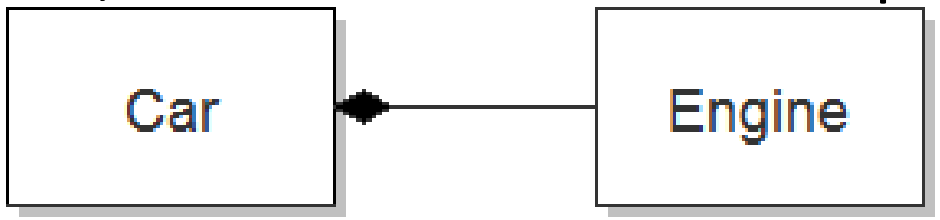
Наследование

“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»,

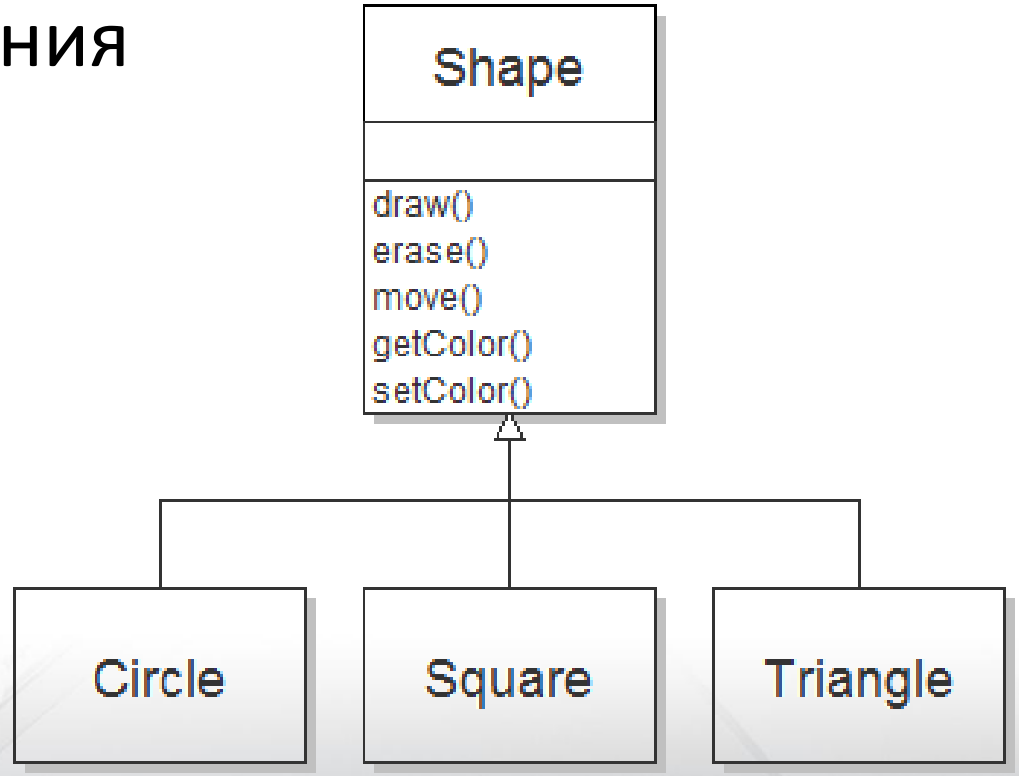
Композиция (Агрегирование)

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

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

Наследование

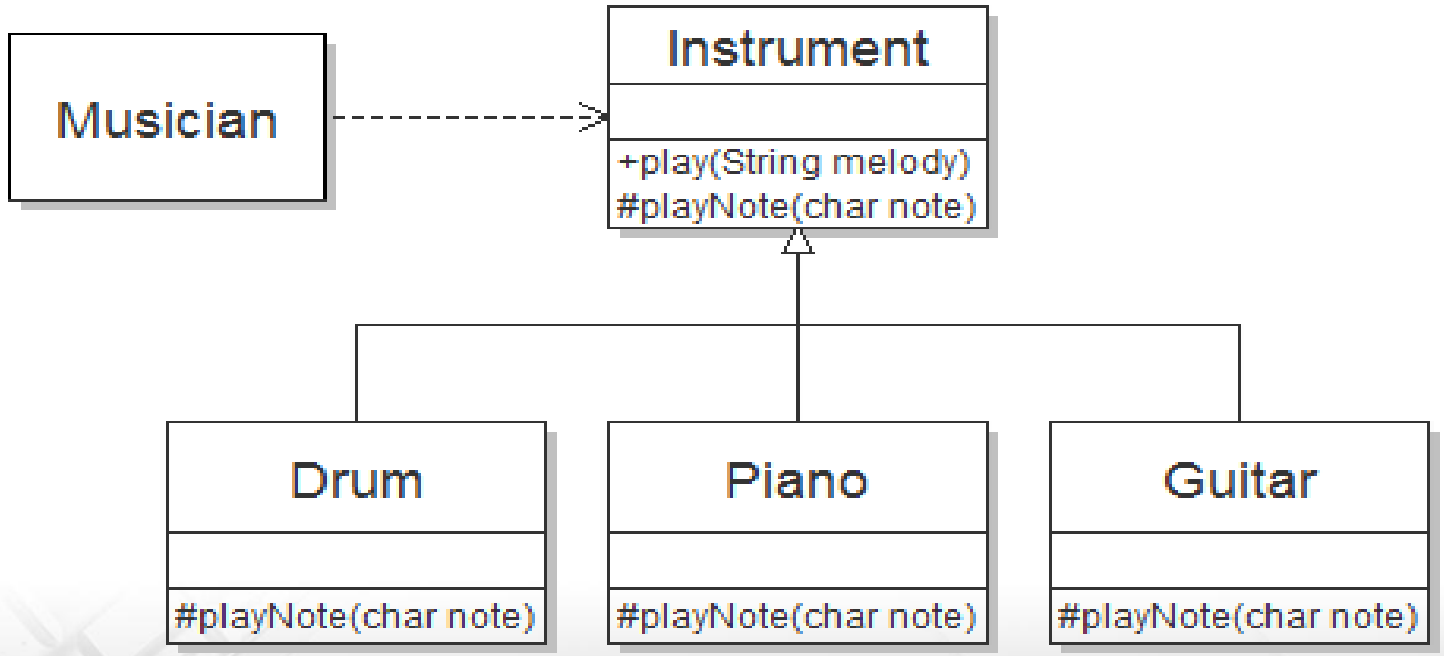
Мы можем взять существующий класс, «клонировать» его, а затем внести дополнения и обновления



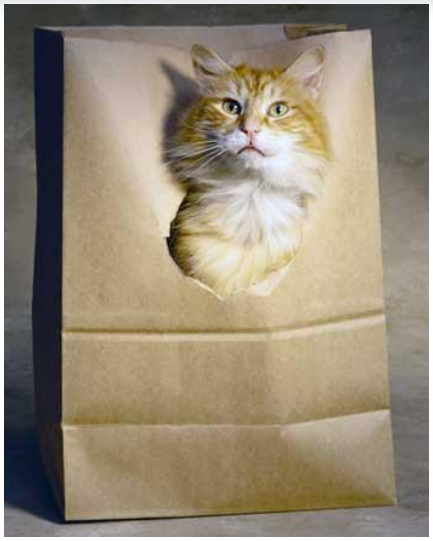
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Полиморфизм

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



Пакеты



Пакет (package) представляют собой набор классов, объединённых по смыслу.

Пакеты обычно вкладываются друг в друга, образуя собой иерархию. В файловой системе такая иерархия выглядит в виде вложенных друг в друга папок.

Типичный путь к пакету выглядит примерно так:
org.apache.commons.collectons.

Пакеты

Чтобы использовать какой-то класс в коде другого класса, надо импортировать его, написав до объявления класса строку

```
import пакет.подпакет.ИмяКласса;
```

Кроме того, если вы используете классы одного пакета часто, вы можете импортировать весь пакет:

```
import пакет.подпакет.*;
```

Это относится ко всем пакетам, кроме java.lang — он импортирован по умолчанию, именно из него были в прошлом примере взяты классы System и String. На самом деле они лежат в некоем jar'e, в каталоге java/lang.

Пакеты решают проблему идентификации, в случае если у вас классы с одинаковыми именами, разложите их по разным пакетам.

Пакеты

Чтобы использовать какой-то класс в коде другого класса, надо импортировать его, написав до объявления класса строку

```
import пакет.подпакет.ИмяКласса;
```

Кроме того, если вы используете классы одного пакета часто, вы можете импортировать весь пакет:

```
import пакет.подпакет.*;
```

Это относится ко всем пакетам, кроме java.lang — он импортирован по умолчанию, именно из него были в прошлом примере взяты классы System и String. На самом деле они лежат в некоем jar'e, в каталоге java/lang.

Пакеты решают проблему идентификации, в случае если у вас классы с одинаковыми именами, разложите их по разным пакетам.

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*.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Модификаторы доступа

- 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

- Модификатор **default** или, как его еще называют, package visible. Переменная или метод, объявленные без модификатора доступны для любого другого класса в том же пакете. Поля в интерфейсе неявно являются public, static, final, а методы в интерфейсе по умолчанию являются public..