НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 1. Introduction to Objects

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

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

The progress of abstraction

- Assembly language is a small abstraction of the underlying machine.
- Many so-called “imperative” languages that followed (such as FORTRAN, BASIC, and C) were abstractions of assembly language
- The object-oriented approach goes a step further by providing tools for the programmer to represent elements in the problem space

The progress of abstraction

- OOP allows you to describe the problem in terms of the problem, rather than in terms of the computer where the solution will run.
- There's still a connection back to the computer:

Each object looks quite a bit like a little computer — it has a state, and it has operations that you can ask it to perform

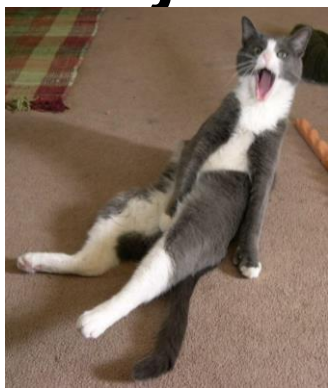
Characteristics of OOP

1. Everything is an object
2. A program is a bunch of objects telling each other what to do by sending messages
3. Each object has its own memory made up of other objects
4. Every object has a type
5. All objects of a particular type can receive the same messages

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

What object is?

An object has state, behavior and identity



This means that an object can have internal data (which gives it state), methods (to produce behavior), and each object can be uniquely distinguished from every other object — to put this in a concrete sense, each object has a unique address in memory

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

An object has an interface

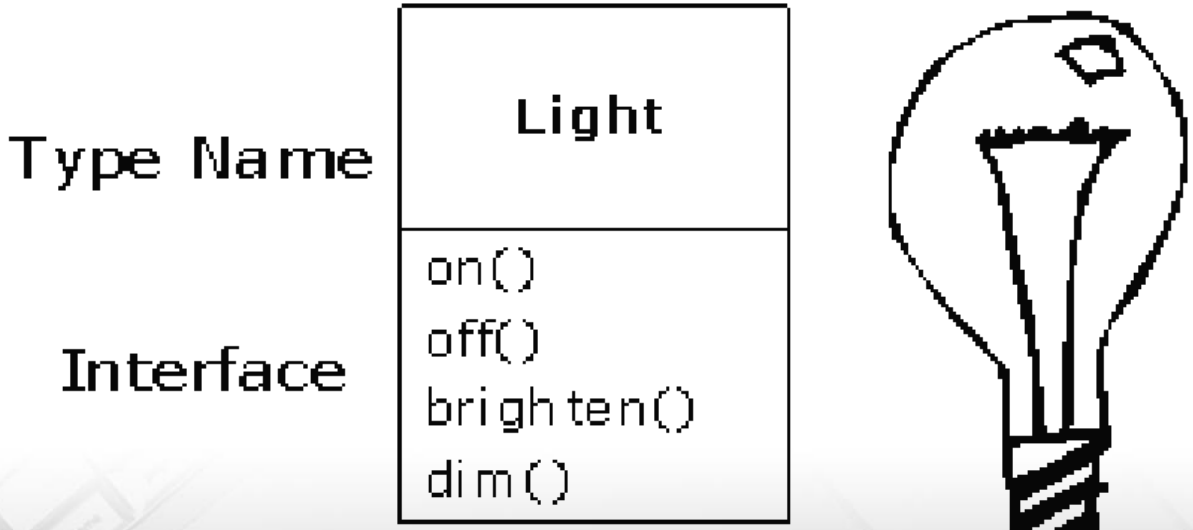
In object-oriented programming we create new data types, but all object-oriented programming languages use the “class” keyword.

When you see the word “type” think “class” and vice versa

Once a class is established, you can make as many objects of that class as you like, and then manipulate those objects as if they are the elements that exist in the problem you are trying to solve.

An object has an interface

Each object can satisfy only certain requests. The requests you can make of an object are defined by its *interface*, and the type is what determines the interface




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

Reusing the implementation

Composition (Aggregation)

"has-a"

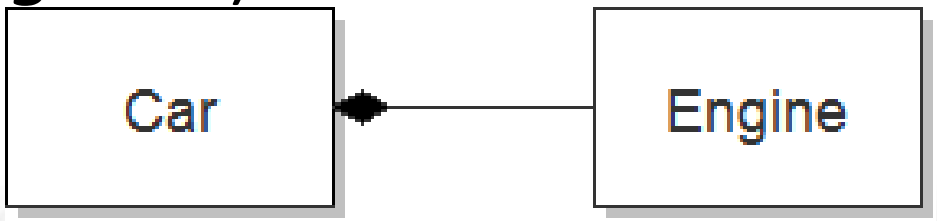
Inheritance

"is-a"

Composition / Aggregation

The simplest way to reuse a class is to just use an object of that class directly, but you can also place an object of that class inside a new class

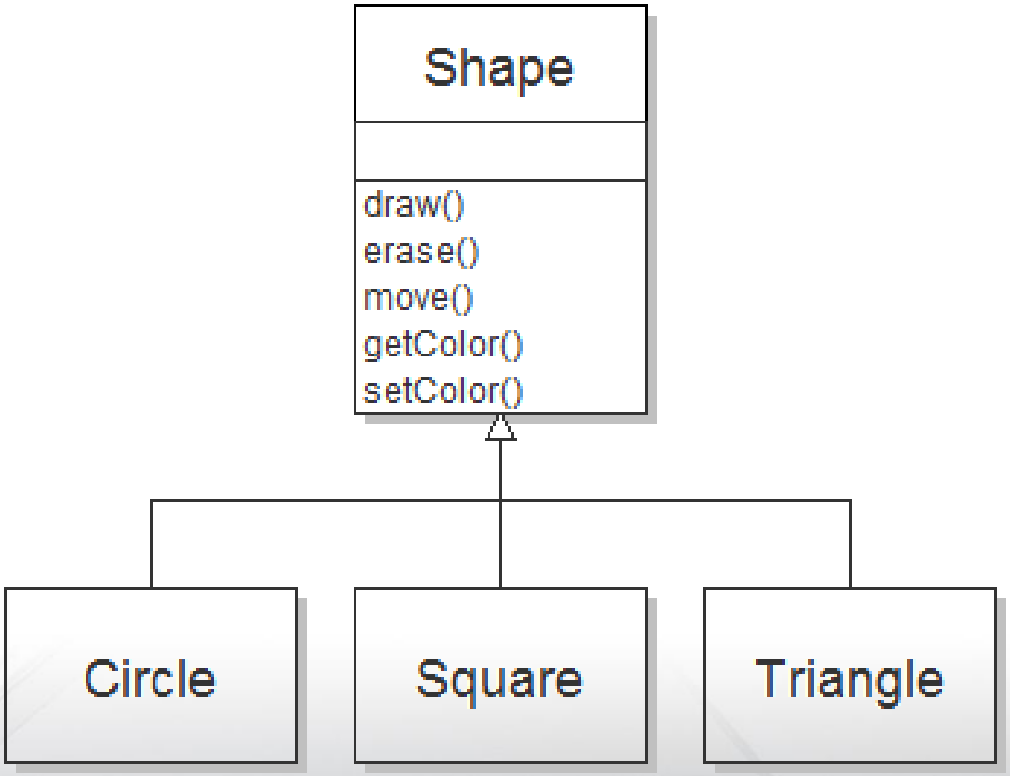
Because you are composing a new class from existing classes, this concept is called ***composition*** (if the composition happens dynamically, it's usually called ***aggregation***).



Composition is often referred to as a "**has-a**" relationship, as in "**A car has an engine**"

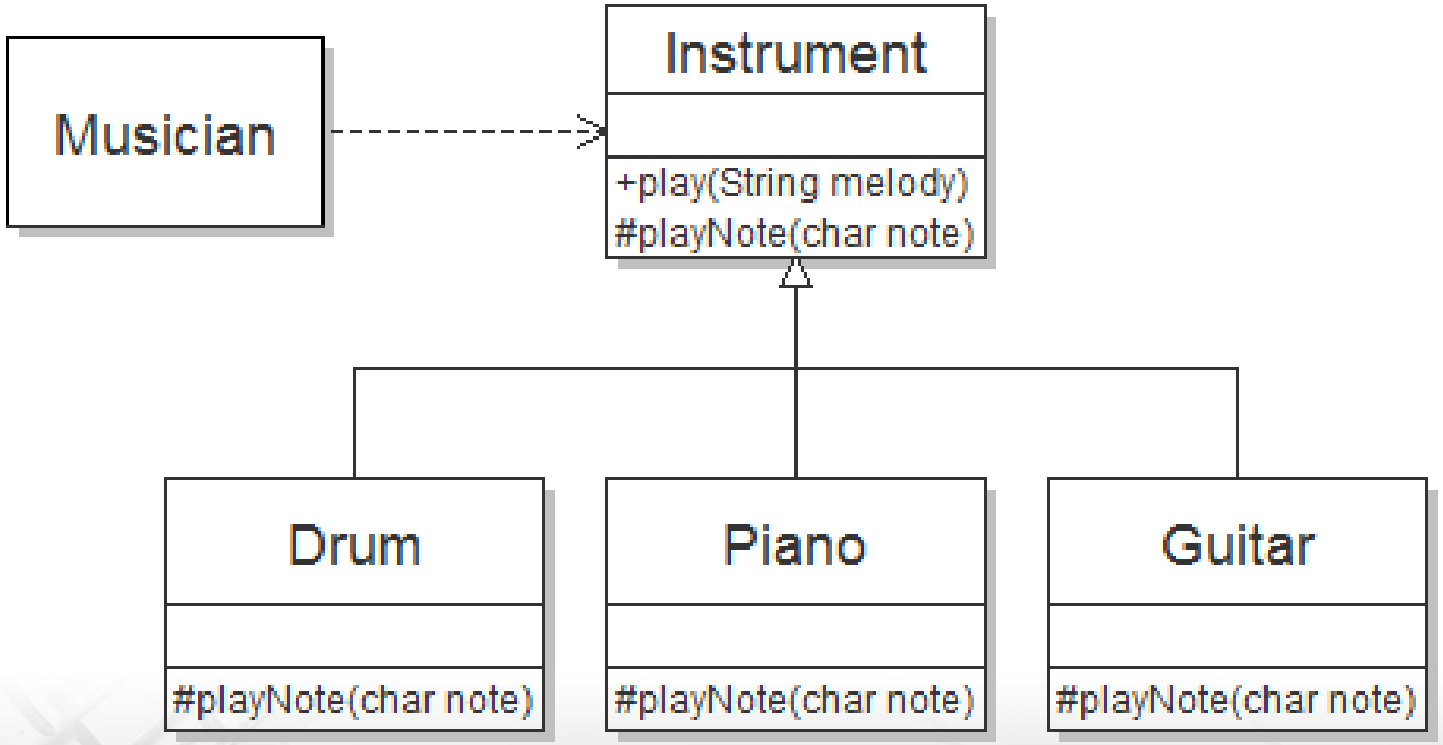
Inheritance

We can take the existing class, clone it, and then make additions and modifications to the clone



Polymorphism

Let’s consider a musician, that uses musical instrument for play



Example

Java Basics

You treat everything as an object, using a single consistent syntax. Although you treat everything as an object, the identifier you manipulate is actually a “reference” to an object

You must create all the objects

When you create a reference, you want to connect it with a new object. You do so, in general, with the **new** operator:

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

Java Basics

Special case: primitive types

Java determines the size of each primitive type.

These sizes don't change from one machine architecture to another as they do in most languages.

This size invariance is one reason Java programs are more portable than programs in most other languages.

Primitive types

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**

Arrays in Java

- Java array is guaranteed to be initialized and cannot be accessed outside of its range.
- The range checking comes at the price of having a small amount of memory overhead on each array as well as verifying the index at run time, but the assumption is that the safety and increased productivity are worth the expense

Arrays in Java

- When you create an array of objects, you are really creating an array of references, and each of those references is automatically initialized to a special value with its own keyword: **null**
- You can also create an array of primitives. Again, the compiler guarantees initialization because it zeroes the memory for that array

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

Access modifiers

- Java provides access specifiers to allow the library creator to say what is available to the client programmer and what is not.
- The levels of access control from “most access” to “least access” are **public**, **protected**, package access (which has no keyword), and **private**.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Access modifiers

For members (fields and methods)

Modifiers	Members
`public`	Accessible everywhere.
`protected`	Accessible by any class in the same package as its class, and accessible only by subclasses of its class in other packages.
default (no modifier)	Only accessible by classes, including subclasses, in the same package as its class (package accessibility).
`private`	Only accessible in its own class and not anywhere else.

For top-level types (Classes, Interfaces, Enums…)

Modifiers	Top-Level Types
default (no modifier)	Accessible in its own package (*package accessibility*)
`public`	Accessible anywhere

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other access modifiers

For top-level types (Classes, Interfaces, Enums…)

Modifiers	Classes	Interfaces	Enum types
`abstract`	A non-`final` class can be declared `abstract`. A class with an `abstract` method must be declared `abstract`. An `abstract` class cannot be instantiated.	Permitted, but redundant.	Not permitted.
`final`	A non-`abstract` class can be declared `final`. A class with a `final` method need not be declared `final`. A `final` class cannot be extended.	Not permitted.	Not permitted.

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Other access modifiers

For members (fields and methods)

Modifiers	Fields	Methods
`static`	Defines a class variable.	Defines a class method.
`final`	Defines a constant.	The method cannot be overridden.
`abstract`	Not applicable.	No method body is defined. Its class must also be designated `abstract`.
`synchronized`	Not applicable.	Only one thread at a time can execute the method.
`native`	Not applicable.	Declares that the method is implemented in another language.
`transient`	The value in the field will not be included when the object is serialized.	Not applicable.
`volatile`	The compiler will not attempt to optimize access to the value in the field.	Not applicable.

Example

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Questions?

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ КОРАБЛЕБУДУВАННЯ ІМЕНІ АДМІРАЛА МАКАРОВА

Object-Oriented Programming in the Java language

Part 1. Introduction to Objects

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