



May 2019, IPT Course
Java Web Debelopment

JWD: OOP, String Processing, Formatting, RegEx, Resources

Trayan Iliev

tiliev@ipproduct.org

<http://ipproduct.org>

Copyright © 2003-2019 IPT - Intellectual
Products & Technologies

About me



Trayan Iliev

- CEO of IPT – Intellectual Products & Technologies
- Oracle® certified programmer 15+ Y
- end-to-end reactive fullstack apps with Java, ES6/7, TypeScript, Angular, React and Vue.js
- 12+ years IT trainer
- Voxxed Days, jPrime, jProfessionals, BGOUG, BGJUG, DEV.BG speaker
- Organizer RoboLearn hackathons and IoT enthusiast (<http://robolearn.org>)

Where to Find the Code?

Java Web Development projects and examples are available @ GitHub:

<https://github.com/iproduct/course-java-web-development>

Agenda for This Session

- ❖ OOP principles – Encapsulation, Inheritance and Polymorphism, Overriding / Overloading
- ❖ String Processing,
- ❖ Data Formatting, Resource Bundles, Regular Expressions
- ❖ java.util & java.math
- ❖ StringTokenizer, Date/Calendar,
- ❖ Locale, Random, Optional, Observable, Observable interface, BigDecimal

Basic Concepts in OOP and OOAD

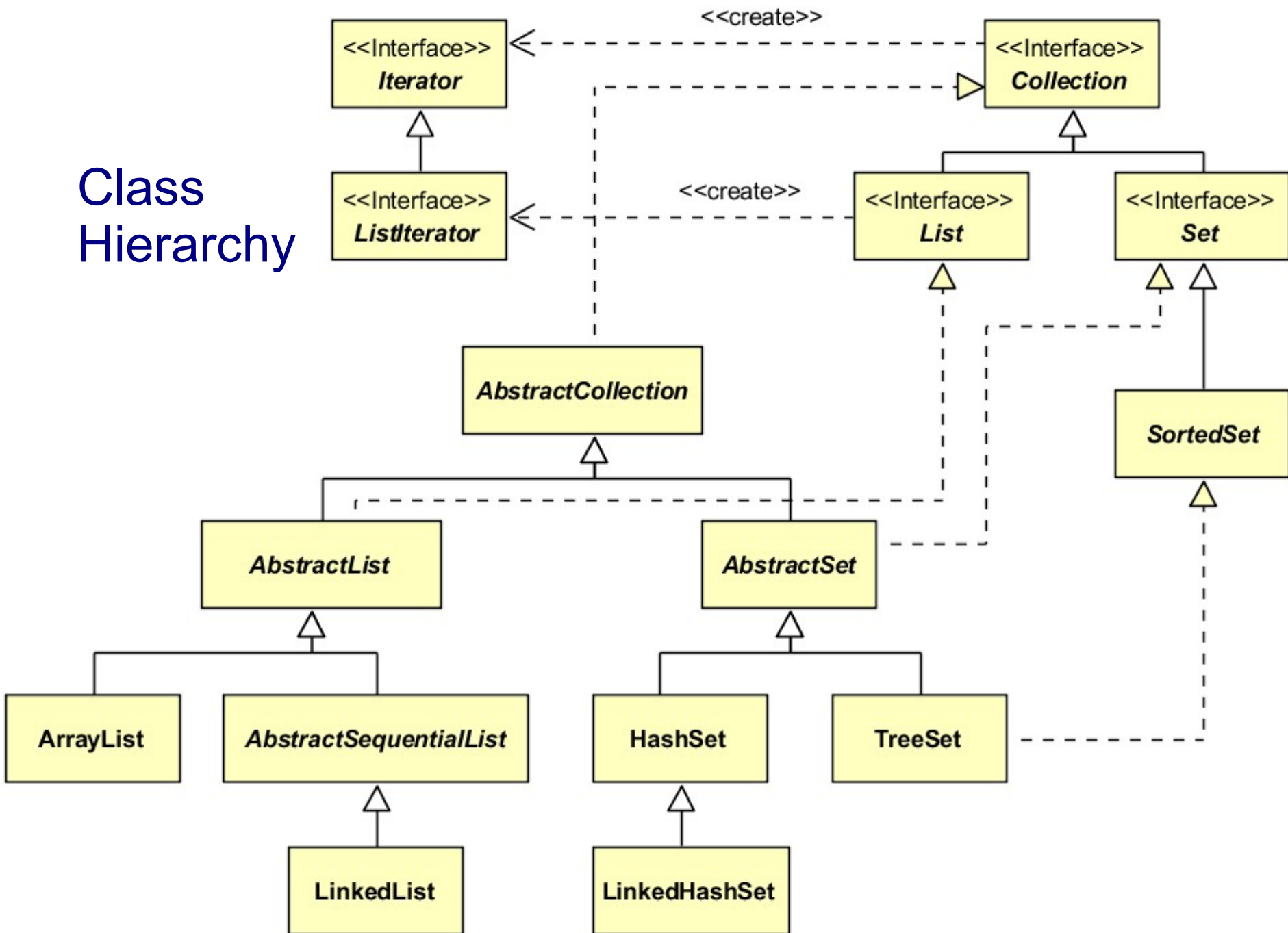
- ❖ interface and implementation – we divide what remains constant (contractual interface) from what we would like to keep our freedom to change (hidden realization of this interface)
- ❖ interface = **public**
- ❖ implementation = **private**
- ❖ This separation allows the system to evolve while maintaining backward compatibility to already implemented solutions, enables parallel development of multiple teams
- ❖ **programming based on contractual interfaces**

Object-Oriented Approach to Programming

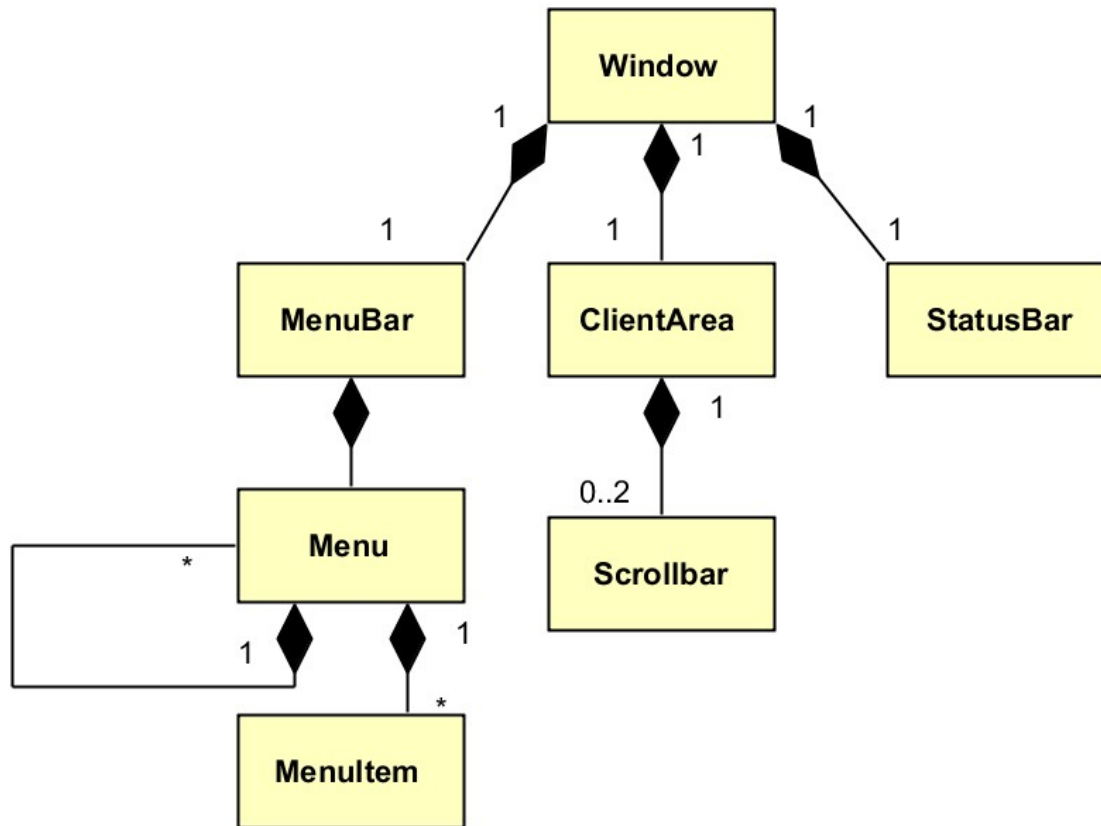
Key elements of the object model [Booch]:

- ❖ **class, object, interface and implementation**
- ❖ **abstraction** – basic distinguishing characteristics of an object
- ❖ **capsulation** – separating the elements of abstraction that make up its structure and behavior - interface and implementation
- ❖ **modularity** – decomposing the system into a plurality of components and loosely connected modules - principle: maximum coherence and the minimum connectivity
- ❖ **hierarchy** – class and object hierarchies

Class Hierarchy



Object Hierarchy



Object-Oriented Approach to Programming

Additional elements of the object model [Booch]:

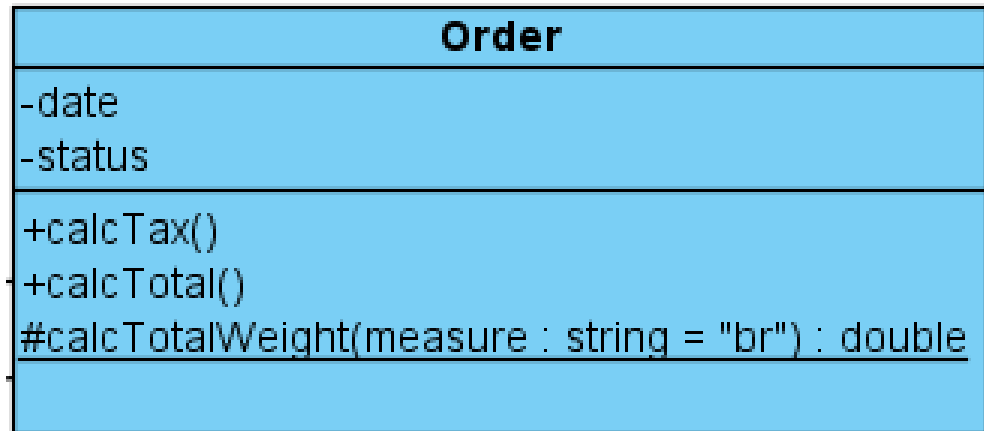
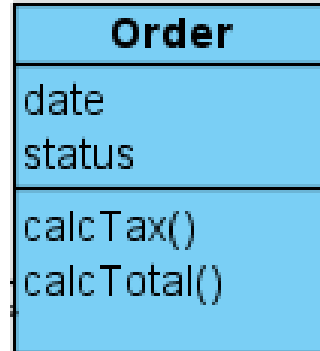
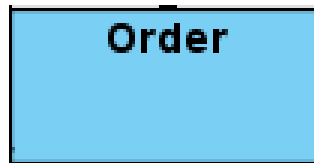
- ❖ **typing** – requirement for the class of an object such that objects of different types can not be replaced (or can in a strictly limited way)
 - static and dynamic binding
 - polymorphism
- ❖ **concurrency** – abstraction and synchronization of processes
- ❖ **length of life** – object-oriented databases

Classes

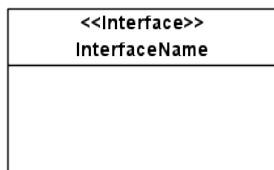
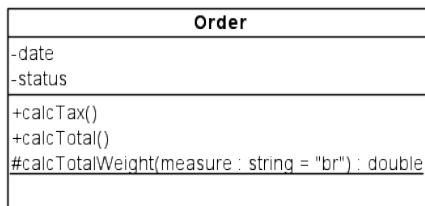
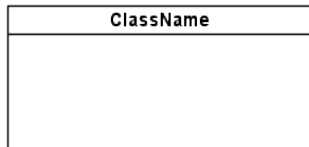
Class – describes a set of objects that share the same specifications of the characteristics (attributes and methods), constraints and semantics

- attributes – instances of properties in UML, they can provide end of association, object *structure*
- operations - behavioral characteristics of a classifier, specifying name, type, parameters and constraints for invoking definitely associated with the operation behavior

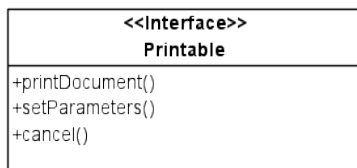
Classes - Graphical Notation in UML



Elements of Class Diagrams

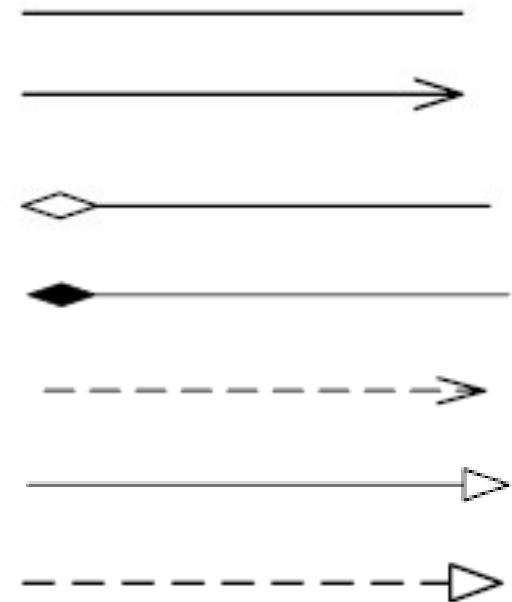


InterfaceName

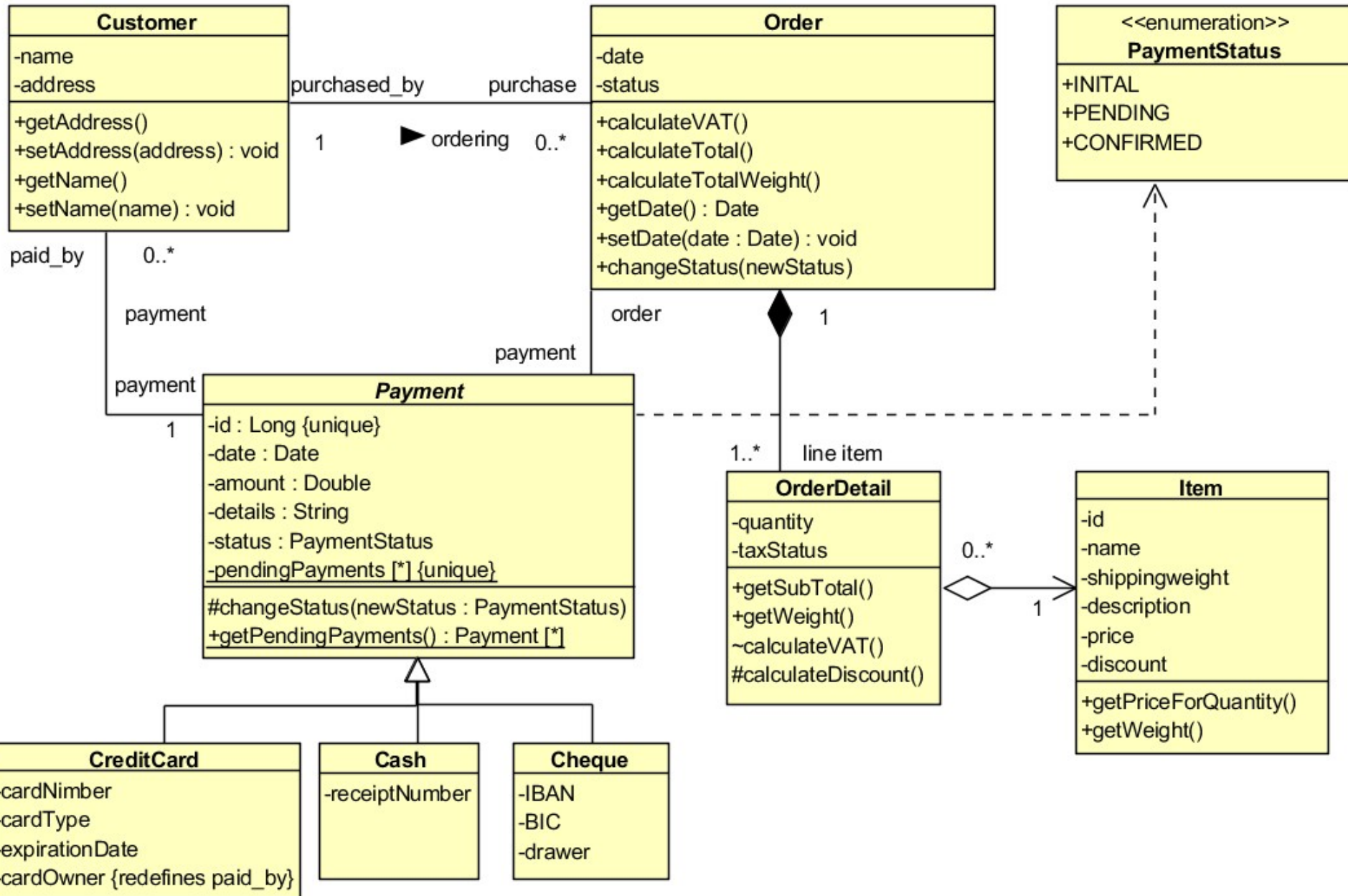


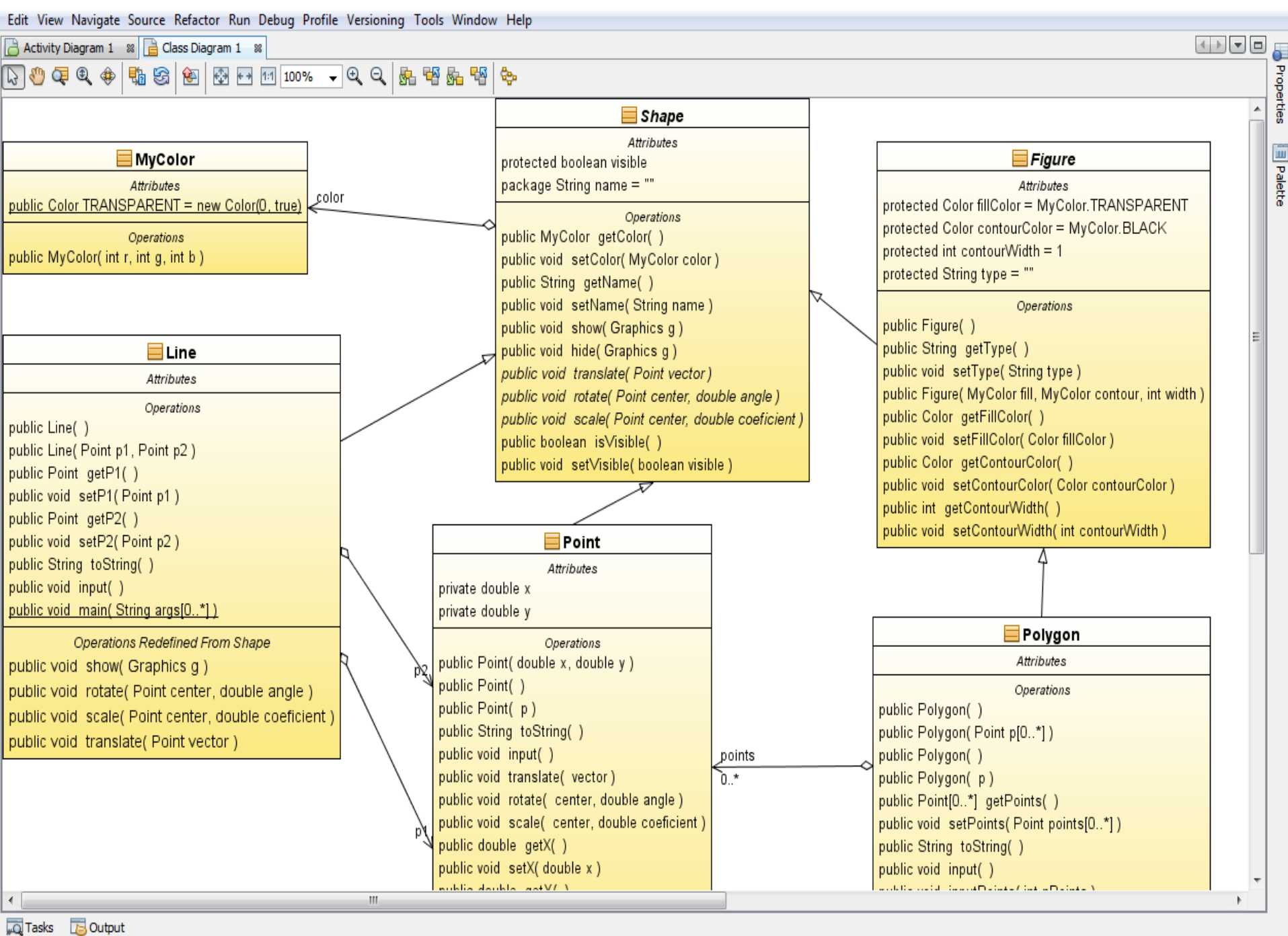
Types of connections:

- association
- aggregation
- composition
- dependence
- generalization
- realization



Class Diagram - 1





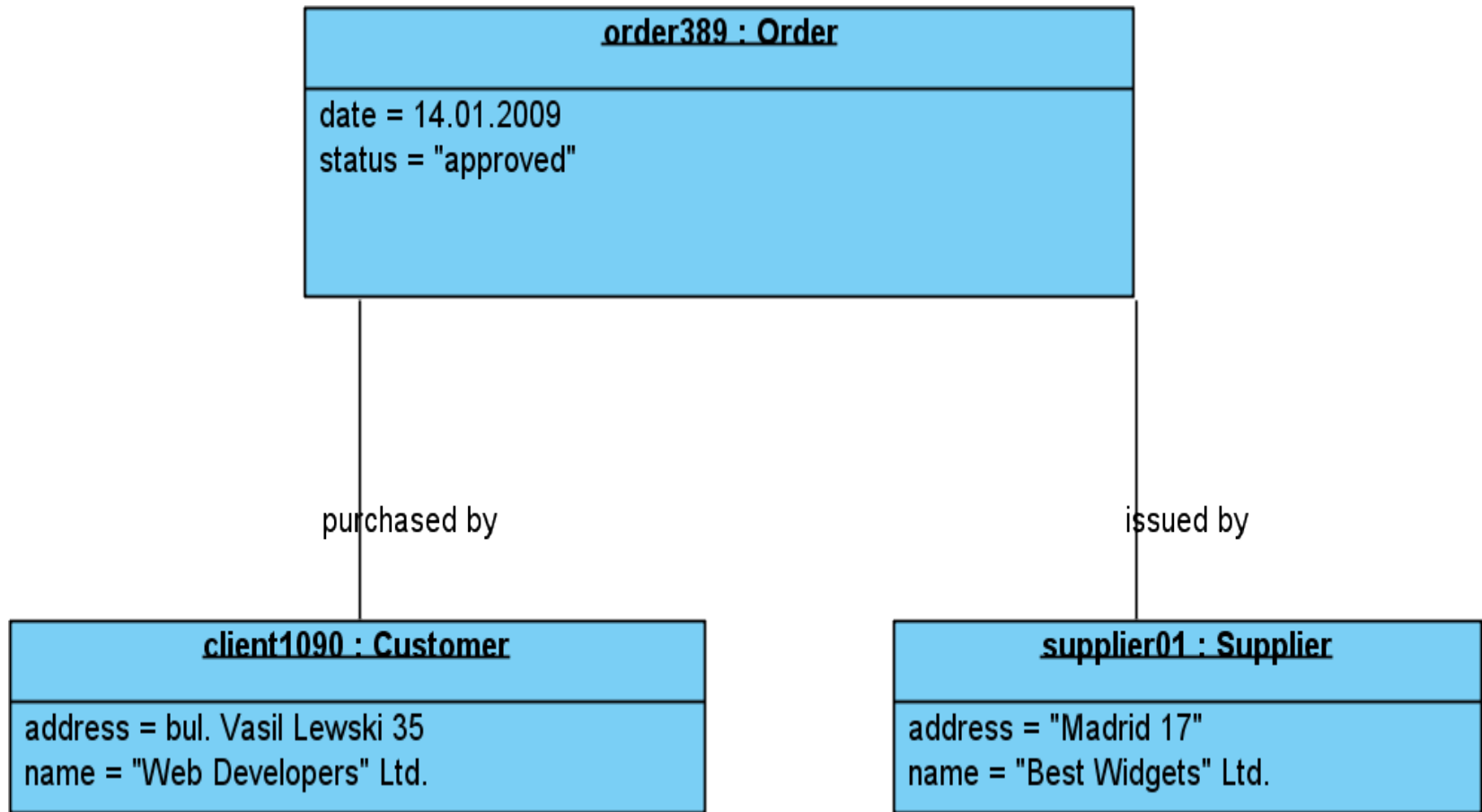
Objects

Instance specification = Object – represents an instance of the modeled system, for example class -> object association -> link, property -> attribute, etc.

- can provide illustration or example of object
- describes the object in a particular moment of time
- may be uncomplete
- Example:

order389 : Order
date = 14.01.2009 status = "approved"

Object Diagram



Analysis Classes Stereotypes

Analysis classes are used in the mapping and analysis of system architecture - they present rather different roles and responsibilities, than specific classes to be realized, and are independent of implementation technology:

- <<controll>> - business logic
- <<entity>> - data
- <<boundary>> - user or system interface



Controlling Class



Class Unit



Border Class

Object Constructors in Java

- ❖ Initialization of objects with constructors
- ❖ **Overloading** of constructors and other methods
- ❖ Default constructors
- ❖ Reference to the current object – **this**

Objects Initialization. Array initialization

- ❖ Initialization in declaration
- ❖ Initialization in constructor
- ❖ „Lazy“ initialization
- ❖ Initialization of static class members
- ❖ One-dimensional and multi-dimensional arrays
- ❖ Array initialization

Strings

- ❖ **String** class provides **immutable** objects – i.e. any operation on the string creates a new object in heap
- ❖ **StringBuilder** – it provides an efficient way from the side of resources to modify the strings, as realize **Reusable Design Pattern: Builder** – for incremental string building (basically with methods **append** and **insert**)
- ❖ Basic **operations** in the class **String**. Formatted output
 - method **format()** and class **Formatter**. Specifiers:

**%[argument_index\$][flags][width]
[.precision]conversion**

Conversion in Type Formatting

- ❖ d – decimal, integral types
- ❖ c – character (unicode)
- ❖ b - boolean
- ❖ s - String
- ❖ f – float, double (with decimal point)
- ❖ e - float, double (scientific notation)
- ❖ x – hexadecimal value of integral types
- ❖ h – hexadecimal hash code

Regular Expressions - I

❖ Symbolic classes:

- `.` Any character (may or may not match line terminators)
- `\d` A digit: `[0-9]`
- `\D` A non-digit: `[^0-9]`
- `\s` A whitespace character: `[\t\n\x0B\f\r]`
- `\S` A non-whitespace character: `[^\s]`
- `\w` A word character: `[a-zA-Z_0-9]`
- `\W` A non-word character: `[^\w]`

Regular Expressions - II

❖ Qualifiers:

- **X?** X, once or not at all
- **X*** X, zero or more times
- **X+** X, one or more times
- **X{n}** X, exactly n times
- **X{n,}** X, at least n times
- **X{n,m}** X, at least n but not more than m times

❖ **Greedy, Reluctant (?) & Possessive (+)** qualifiers

❖ **Capturing Group - (X)**

Regular Expressions - III

❖ Class **Pattern** – basic methods:

- **public static Pattern compile(String regex)**
- **public Matcher matcher(CharSequence input)**
- **public static boolean matches(String regex, CharSequence input)**
- **public String[] split(CharSequence input, int limit)**

❖ Class **Matcher** – basic methods:

- **public boolean matches()**
- **public boolean lookingAt()**
- **public boolean find(int start)**
- **public int groupCount()** и **public String group(int group)**

Packages and Access Specifiers

- ❖ Packages and directories
- ❖ Importing packages – import
- ❖ Access specifiers
 - **public**
 - **private**
 - **protected**
 - **Friendly access** – by default within the package

Reusing Classes

- ❖ Advantages of code reuse
- ❖ Ways of implementation:
 - Objects composition
 - Inheritance of classes (object types)
- ❖ Building complex objects by composition
- ❖ Initializing the references:
 - on declaration of the site
 - in the constructor
 - before using (lazy initialization)

Class Inheritance - I

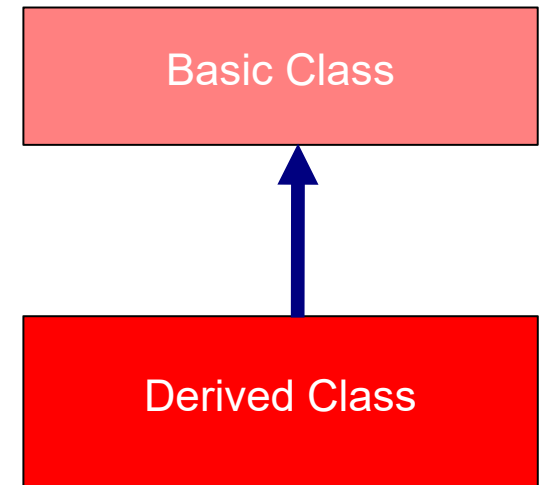
- ❖ Inheritance realization in Java™ language
 - Keyword **extends**
 - Keyword **super**
- ❖ Initialization of objects inheritance:
 - 1) base class; 2) inherited class
 - Calling the default constructors
 - Calling constructors with arguments
- ❖ Combining composition and inheritance

Class Inheritance - II

- ❖ Clearing of objects – realization in Java™
- ❖ Overloading and overriding methods of base class in derived classes
- ❖ When to use composition and when inheritance?
 - Do we need the interface of the base class?
 - Connection Type - „there is“ and „it is“?

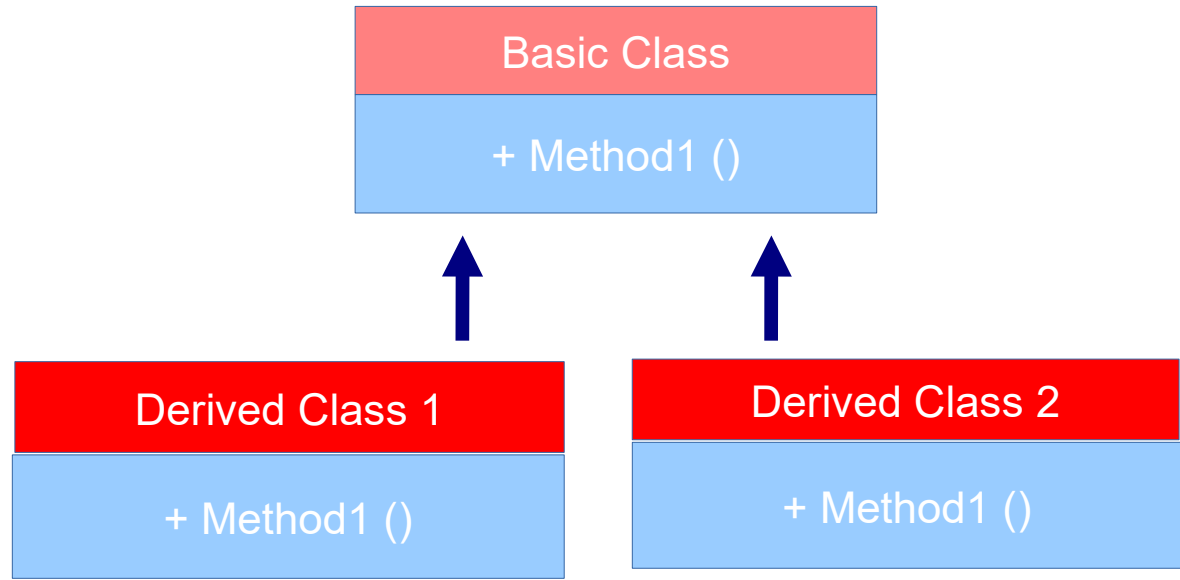
Class Inheritance - III

- ❖ Protected methods
- ❖ Upcasting
- ❖ Keyword final
 - Final data – defining constants
 - simple data type
 - objects
 - empty fields
 - arguments
 - Final methods
 - Final classes



Polymorphism - I

❖ Upcasting



- ❖ Abstract methods and classes – abstract
- ❖ Order of constructor calls
- ❖ Inheritance and expansion

Polymorphism - II

- ❖ **Polymorphism** – by default, unless the method is declared as static or final (private methods become automatically final)
- ❖ When constructing objects with inheritance each object cares about its attributes and **delegate initialization of parental attributes on parental constructor or method**
- ❖ Using polymorphic methods in constructor
- ❖ **Covariance** types of return (from Java SE 5)
- ❖ Composition <-> Inheritance - **State Design Pattern**

Interfaces and Multiple Inheritance

- ❖ Interfaces – keywords: **interface**, **implements**
- ❖ Multiple inheritance in Java
- ❖ Interface expansion through inheritance
- ❖ Constants (static final)
- ❖ Interface incorporation

Advantages of Using Interfaces

- ❖ **Interfaces** cleanly separate requirements type of the object from many possible implementations and make our code more universal and usable
- ❖ **Reusable Design Pattern: Adapter** – It allows to adapt existing realization interface that is required in our application
- ❖ **Inheritance (expansion) of interfaces**
- ❖ **Reusable Design Pattern: Factory Method** – creating reusable client code, isolated from the specifics of the particular server implementation

Inner Classes - I

- ❖ **Inner Classes** group logically related classes and control their visibility
- ❖ **Closures** – internal class has a constant connection to containing outside class and can access all its attributes and even final arguments and local variables (if defined in the method or block)
- ❖ Inner classes can be **anonymous** if used once in the program. Construction.
- ❖ Reference to the object from an external class - **.this** and creating an object from internal class in the context of containing object of the outer class - **.new**

Inner Classes - II

❖ Inner Classes

- defined in an external class
- defined in method
- defined in a block of operators
- access to the attributes of the outer class and to the arguments of the method which are defined in

❖ Anonymous inner classes

- realizing public interface
- inheriting class
- instance initialization
- static inner classes

Thank's for Your Attention!



Trayan Iliev

**CEO of IPT – Intellectual Products
& Technologies**

<http://iproduct.org/>

<http://robolearn.org/>

<https://github.com/iproduct>

<https://twitter.com/trayaniliev>

<https://www.facebook.com/IPT.EACAD>

<https://plus.google.com/+IproductOrg>