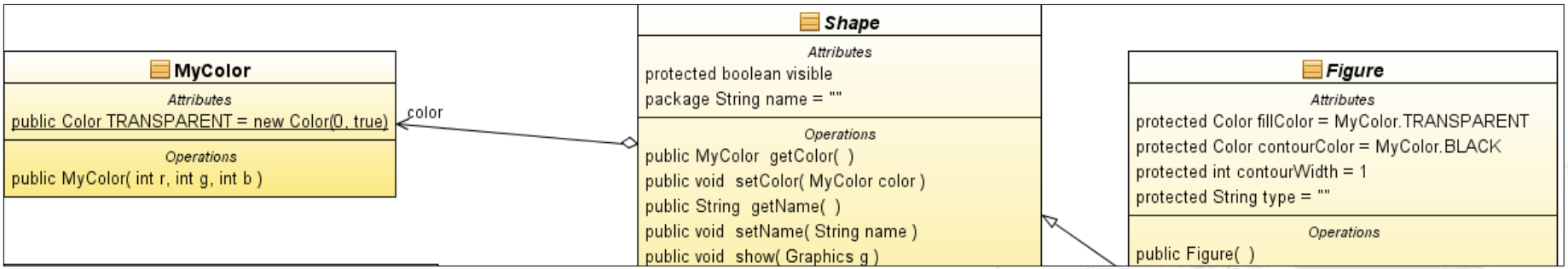
Програмиране на езика Java™



Траян Илиев

IPT – Intellectual Products & Technologies
e-mail: tiliev@iproduct.org
web: http://www.iproduct.org

Развитие на подходите за моделиране и разработка на софтуер

- Компютрите като "велосипеди за ума" - Стив Джобс
- С нарастването на сложността на задачите се появява необходимост от по-съвършени подходи за справяне с тази сложност – източници на сложност [Booch]:
  - сложност на проблемната област
  - постоянна промяна на изискванията на бизнеса движеща се мишена
  - трудността на управление на процеса на разработка
  - недостатъчно развити стандарти позволяващи асемблиране на системите от готови компоненти
  - проблеми свързани с дискретните системи и предсказуемостта на тяхното поведение

Развитие на подходите за моделиране и разработка на софтуер

Пет атрибута на сложната система [Booch,Courtois,Simon,Ando]

- сложност често приема формата на йерархия – т.е. сложната система може да се декомпозира на по-прости подсистеми
- изборът на това докъде да декомпозираме е относително произволен и зависи от целите на разработчика
- вътрешно-компонентните връзки са по-здрави от връзките между отделните компоненти – високочестотна и ниско-честотна комуникационна динамика
- йерархичните системи често са композирани от малък брой различни примитивни части в разнообразни комбинации
- работещата сложна система винаги еволюира от работеща проста система

Развитие на подходите за моделиране и разработка на софтуер

- Поколения алгоритмични езици
  - Език от нулево поколение: машинен език
  - Алгоритмични езици от първо поколение: Асемблер
  - Алгоритмични езици от второ поколение: Кобол, Фортран, Бейсик ...
  - Алгоритмични езици от трето поколение
- структурно програмиране: C, Paskal, ...
- обектно-ориентирано програмиране: Smalltalk, C++ C#, Java, ...

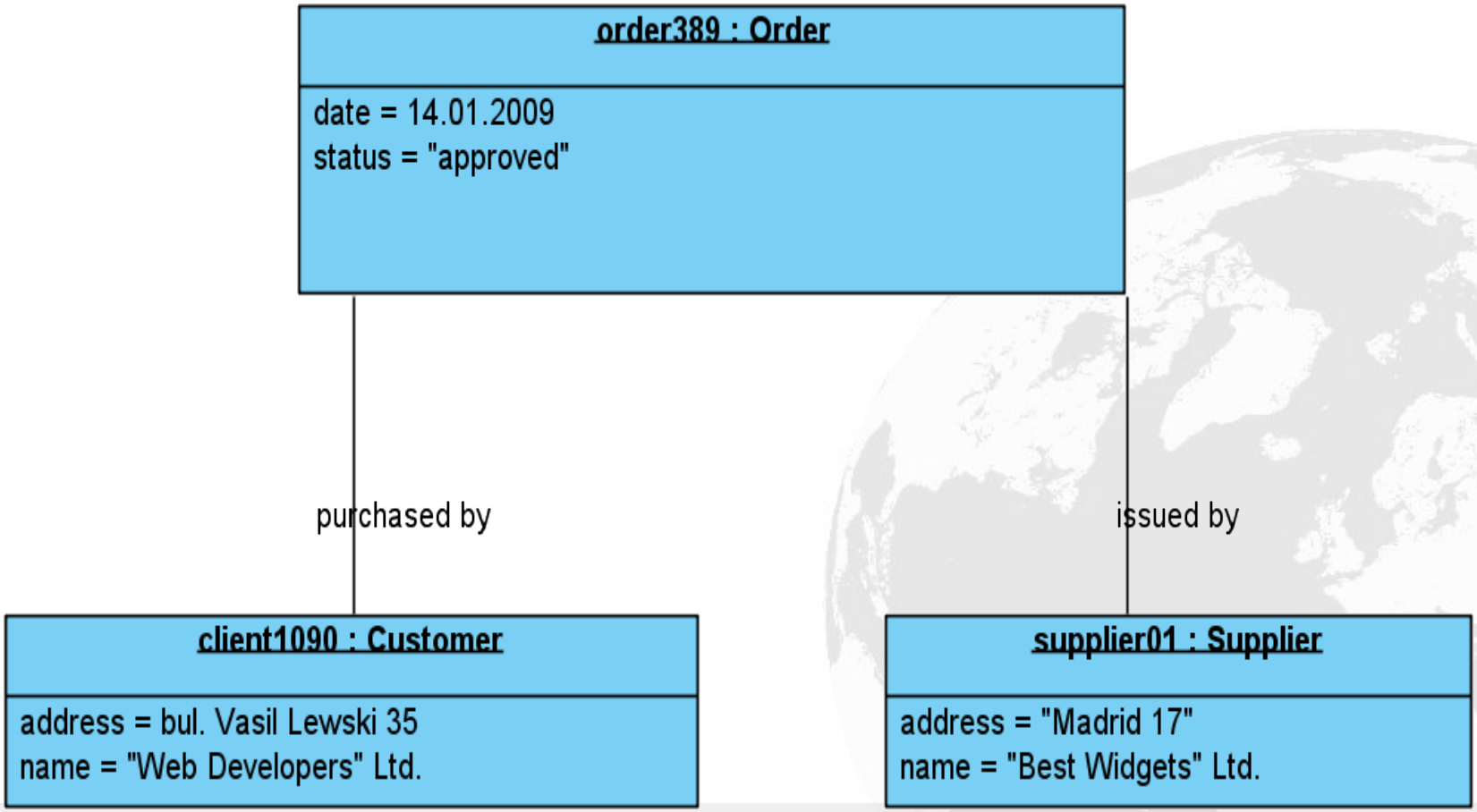
Основни понятия при ООП и ООАД

- Обект:
  - "единици, които комбинират свойствата на процедури и данни, тъй като те изпълняват изчисления и запазват локално състояние" - Stefik и Bobrow
  - "... при обектното моделиране ударението е поставено върху характеризирането на компонентите на физическата или абстрактната ситема, която се моделира чрез програмна система. Обектите имат известен интегритет, който не би трябвало и в същност не може да бъде нарушаван. Един обект може да си променя състоянието или да бъде в релация спрямо други обекти само по начини подходящи за този обект." - Jones

Основни понятия при ООП и ООАД

- Клас – множество от обекти, които споделят обща структура, поведение и възможни връзки с обекти от други класове = тип на обектите
  - структура = атрибути, свойства, член променливи
  - поведение = методи, операции, член функции, съобщения
  - връзки между класовете: асоциация, наследяване / генерализация, агрегация, композиция, използване, инстанциране / метаклас
- Обектите се явяват инстанции на класа, който имат в добавка: 1) собствено състояние и 2) уникален идентификатор

Диаграма на обекти



Основни понятия при ООП и ООАД

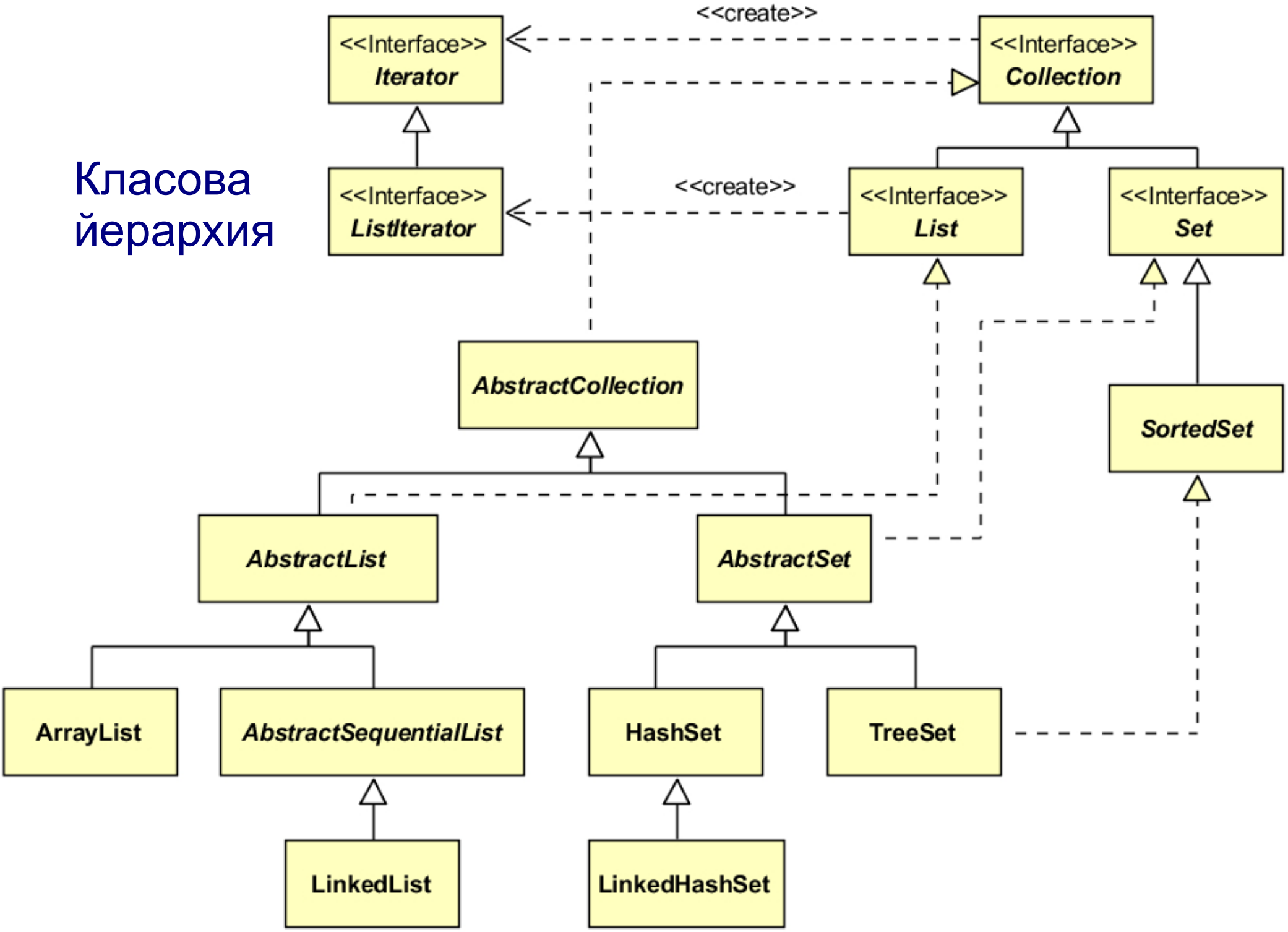
- интерфейс и реализация – разделяме това което остава постоянно (договорен интерфейс) от това което бихме искали да запазим свободата да променяме (скрита реализация на този интерфейс)
- интерфейс = public
- реализация = private
- това разделяне дава възможност системата да еволюира като запазва обратна съвместимост в вече реализираните решения, дава възможност за паралелна разработка от множество екипи - "програмиране базирано на договорени интерфейси"

Обектно-ориентиран подход за програмиране

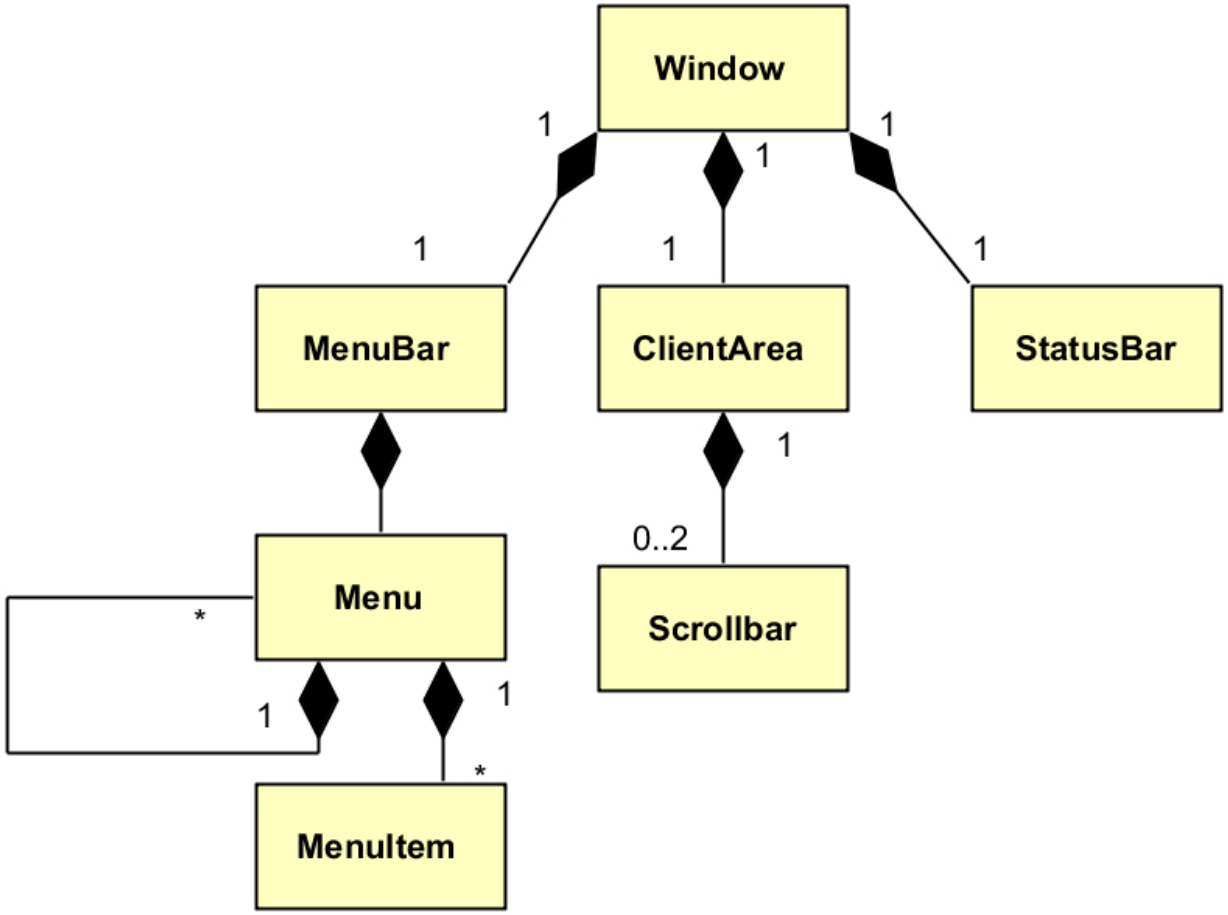
Основни елементи на обектния модел [Booch]:

- клас, обект, интерфейс и реализация
- абстракция – основни, отличителни х-ки на един обект
- капсулация – отделяне елементите на абстракцията които изграждат нейната структура и поведение – интефейс и реализация
- модулност – декомпозиране на системата на множество от компоненти и слабо свързани модули – принцип: максимална кохерентност и минимална свързаност
- йерархичност – класова и обектна йерархии

Класова йерархия



Обектна йерархия



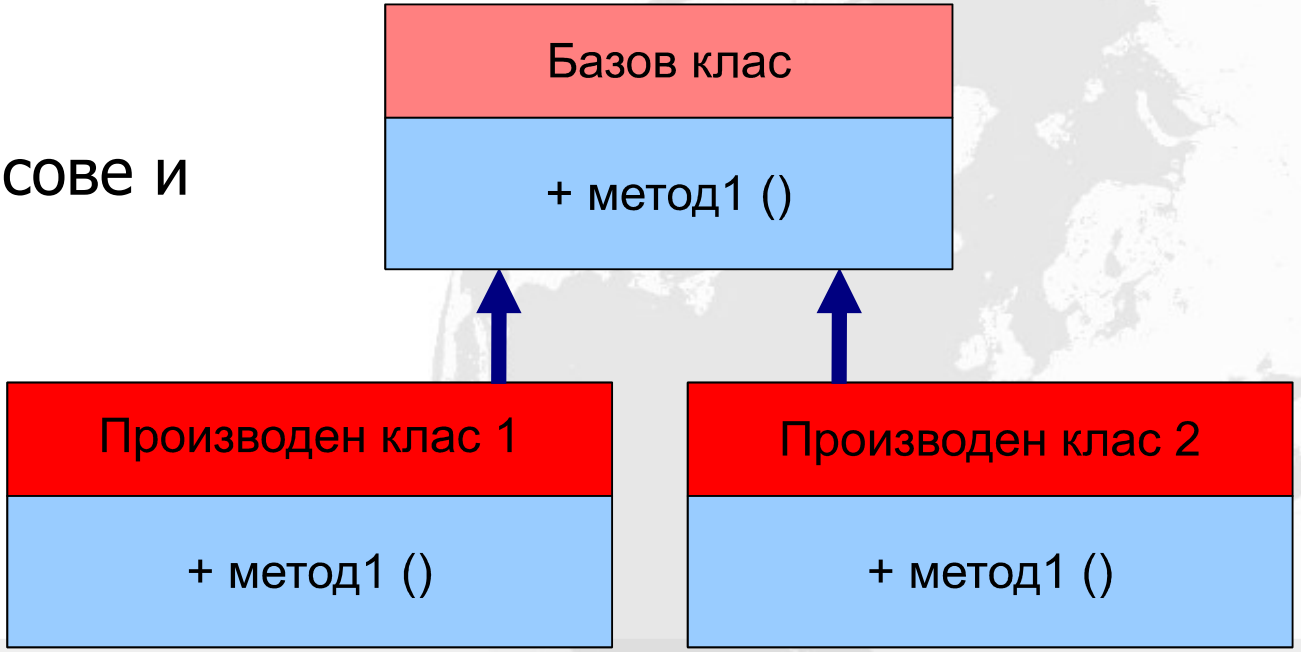
Обектно-ориентиран подход за програмиране

Допълнителни елементи на обектния модел [Booch]:

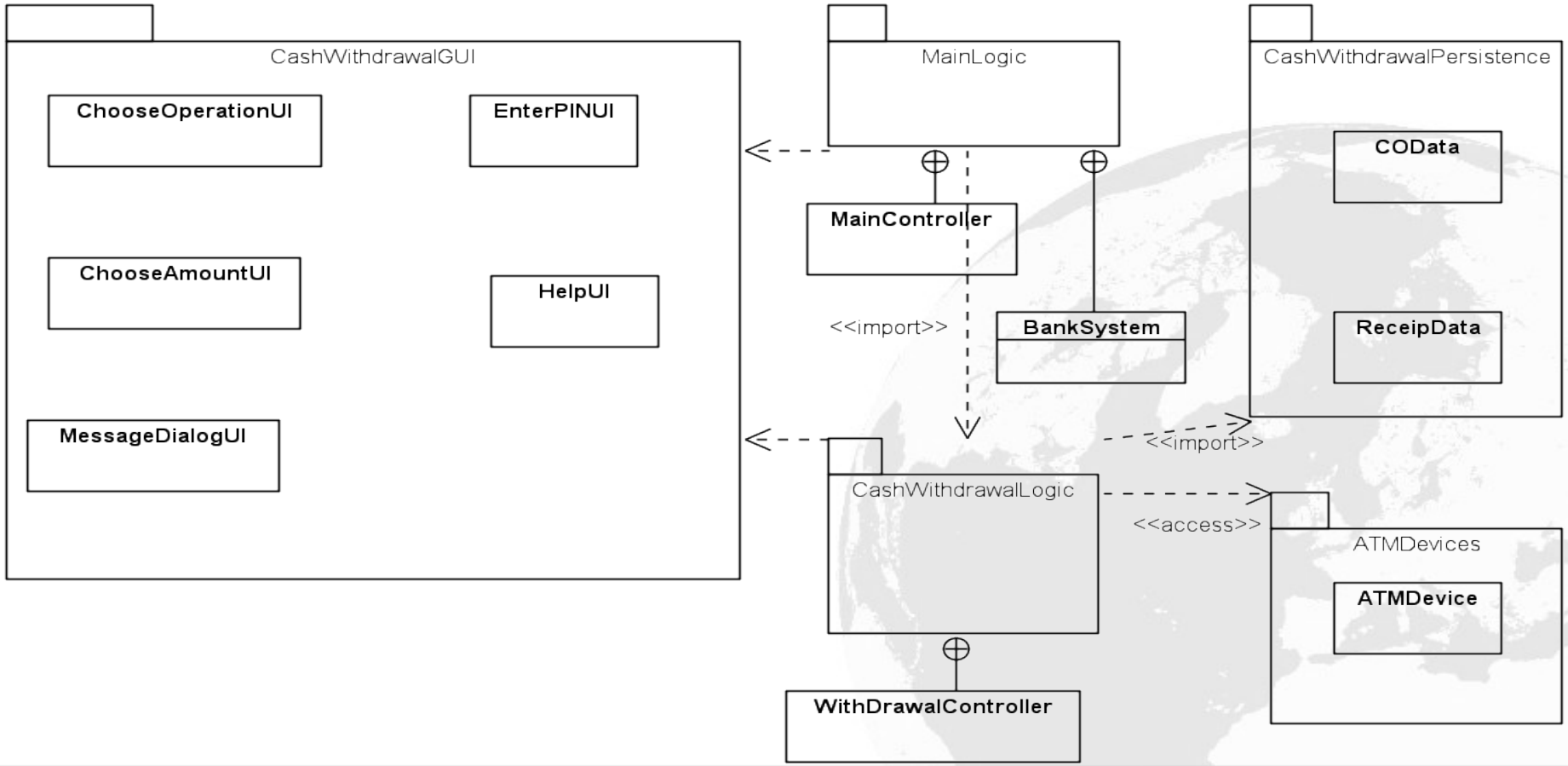
- типизиране – изискване за класа на един обект, такова че обекти от различни типове да не могат да бъдат заменяни (или да могат по строго ограничен начин)
  - статично и динамично свързване
  - полиморфизъм
- конкурентност – абстракция и синхронизация на процеси
- продължителност на съществуването – обектно-ориентирани бази от данни

Подходи за справяне със сложността на софтуера

- Конструиране на обекти от по-прости обекти - композиция
- Наследяване и многократно използване на интерфейса
- Полиморфизъм
- Абстрактни класове и интерфейси
- Контейнери



Диаграма на пакети



Употреба на Unified Modeling Language UML®

Увод в моделирането:

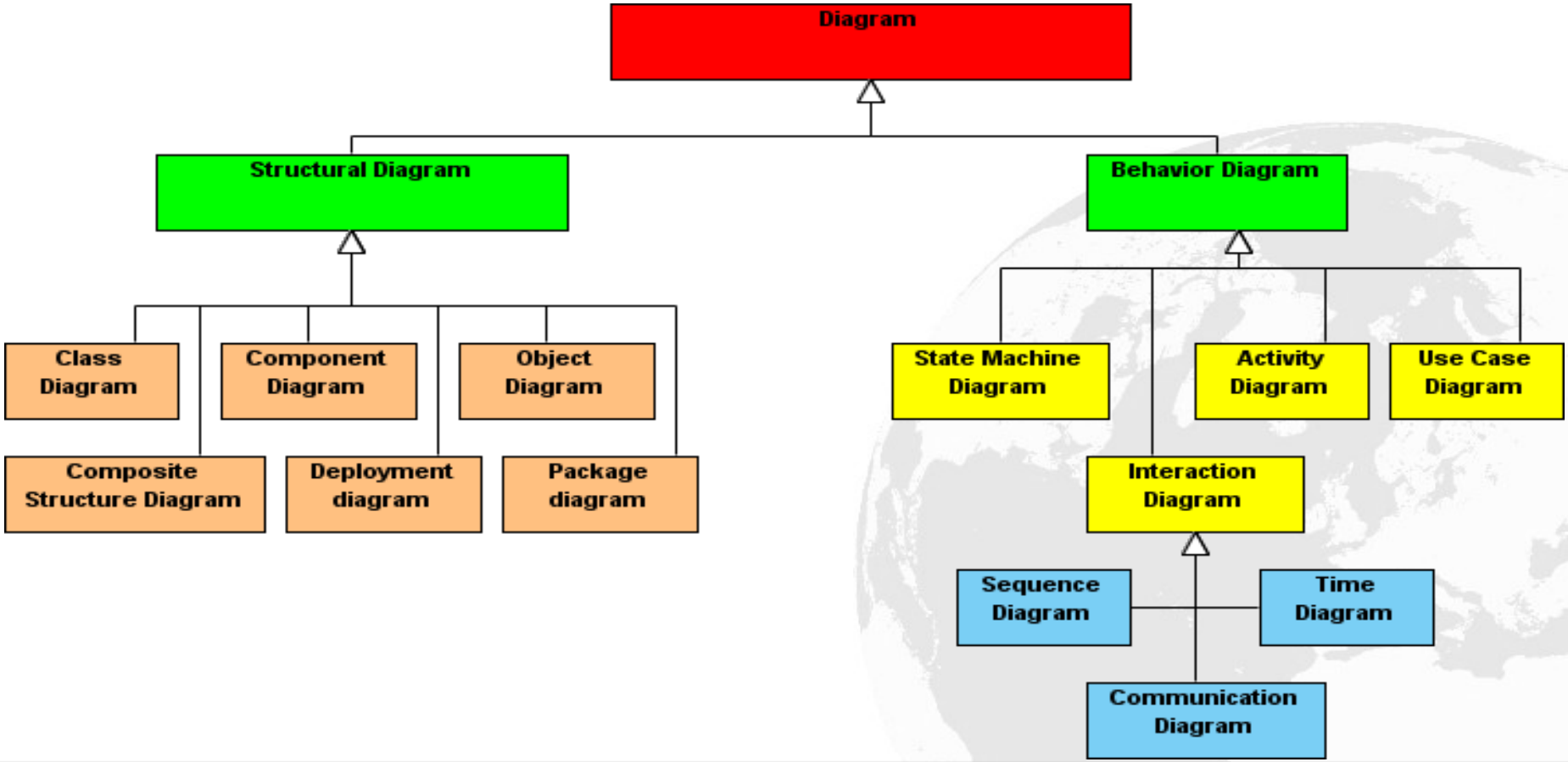
- Моделирането е процесът на дизайн на софтуерното приложение преди кодирането. Моделирането е съществена част от големите софтуерни проекти, и е полезно за средни и дори малки проекти.
- Чрез моделирането се осигуряват завършеност и коректност на бизнес функционалността, задоволяване на нуждите на крайния потребител, и покриване на изискванията за мащабируемост, сигурност, разширяемост и други преди кодирането.
- Моделите ни помагат да работим на по-високо ниво на абстрактност – Semantic Zooming.

Използване на UML®

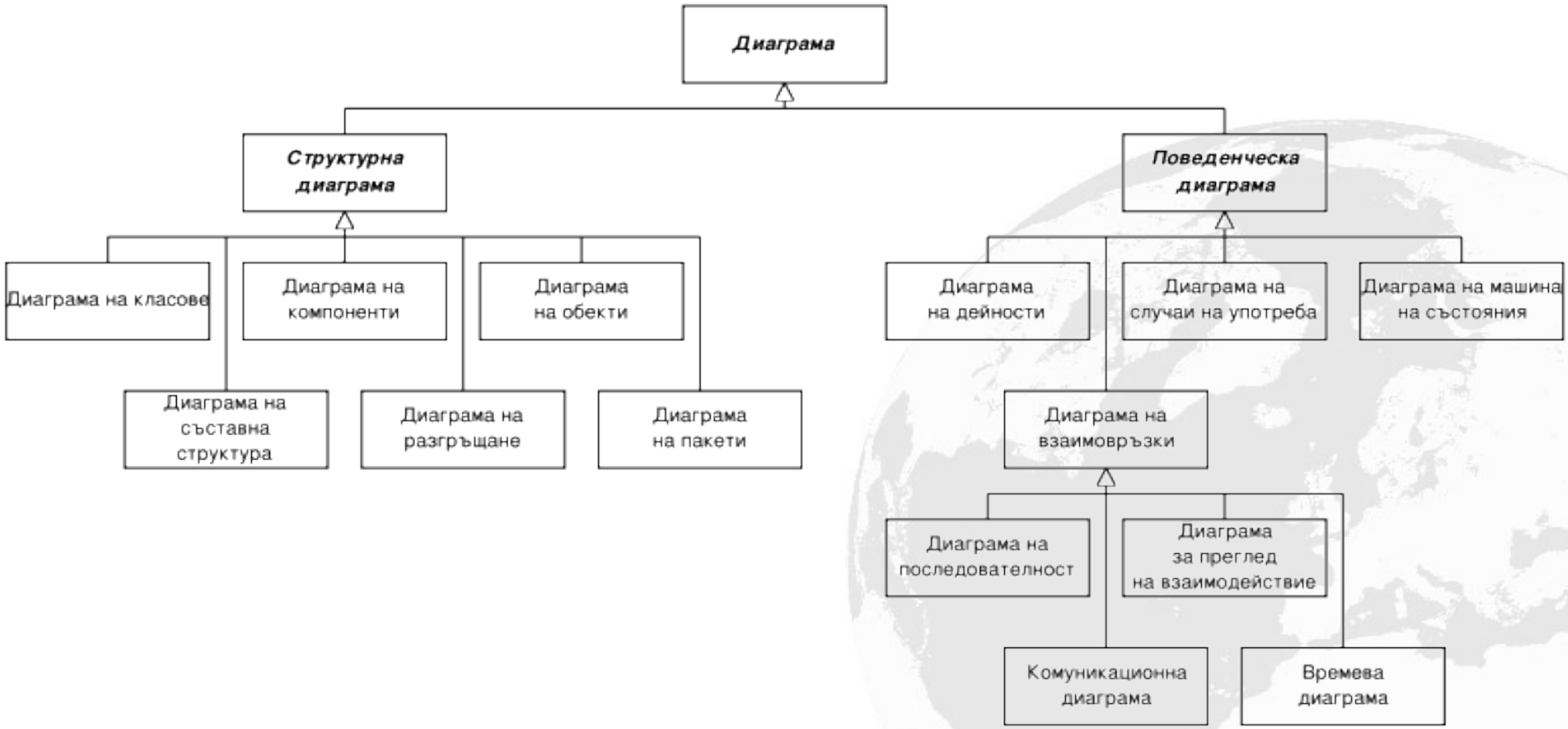
Три основни типа употреба на UML [Fowler, 04]:

- UML като скица
  - forward engineering
  - reverse engineering
  - round-trip engineering
- UML като подробен план (blueprint)
- Изпълним UML (executable UML)
  - UML като език за програмиране
  - Model Driven Architecture (MDA)

UML® диаграми



UML® диаграми (бълг. - Wikipedia)

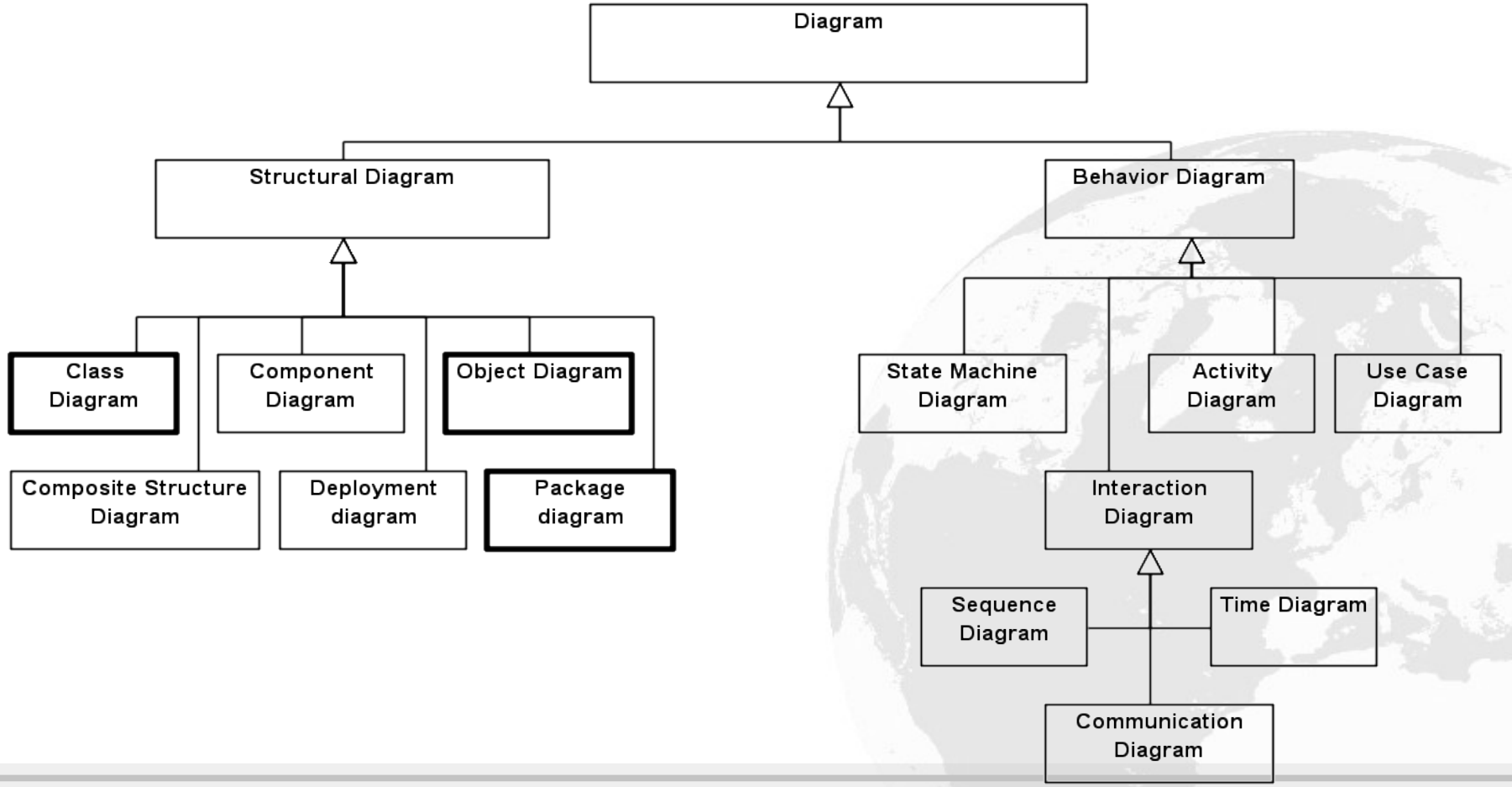


Гледни точки (разрези) на модела

Една софтуерна система може да бъде разгледана от различни перспективи:

- Изисквания към софтуера – случаи на употреба: Use Case D., Activity D.
- Логическа – структура и поведение: Class D., Package D., Object D, State D., Activity D., Sequence D., Communication D.
- Компонентна перспектива: Component D.
- Конкурентност на процесите: State D., Sequence D., Communication D., Activity D., Component D., Deployment D.
- Внедряване: Deployment D.

Структурни диаграми в UML



Структурни диаграми в UML

Структурни диаграми:

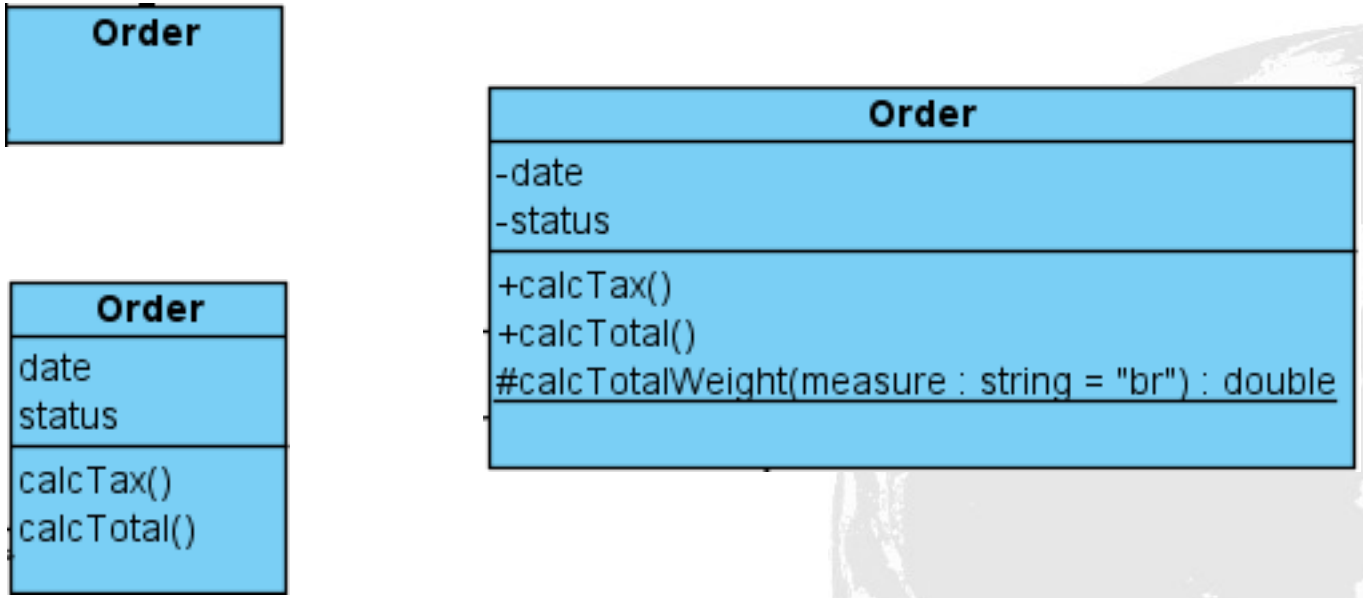
- Класова диаграма (Class Diagram),
- Обектна диаграма (Object Diagram),
- Диаграма на пакетите (Package Diagram),
- Диаграма на компонентите (Component Diagram)
- Диаграма на съставна структура *(Composite Structure Diagram)*
- Диаграма на разгръщане *(Deployment Diagram)*

Класове

Клас (Class) – описва множество от обекти, които споделят едни и същи спецификации на характеристики (атрибути и методи), ограничения и семантика

- атрибути – инстанции на свойства в UML, могат да представят край на асоциация, *структура* на обектите
- операции – *поведенчески* характеристики на класификатор, които специфицират името, типа, параметрите и ограниченията за извикване на определено, асоциирано с операцията поведение

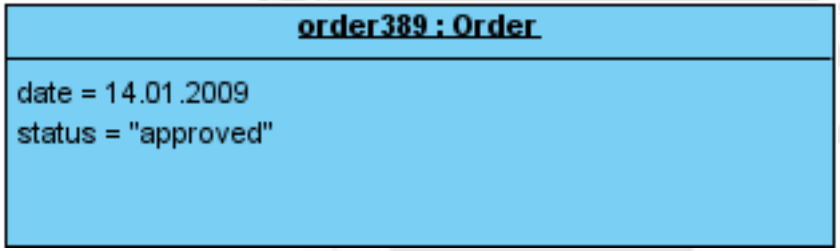
Класове – графична нотация в UML



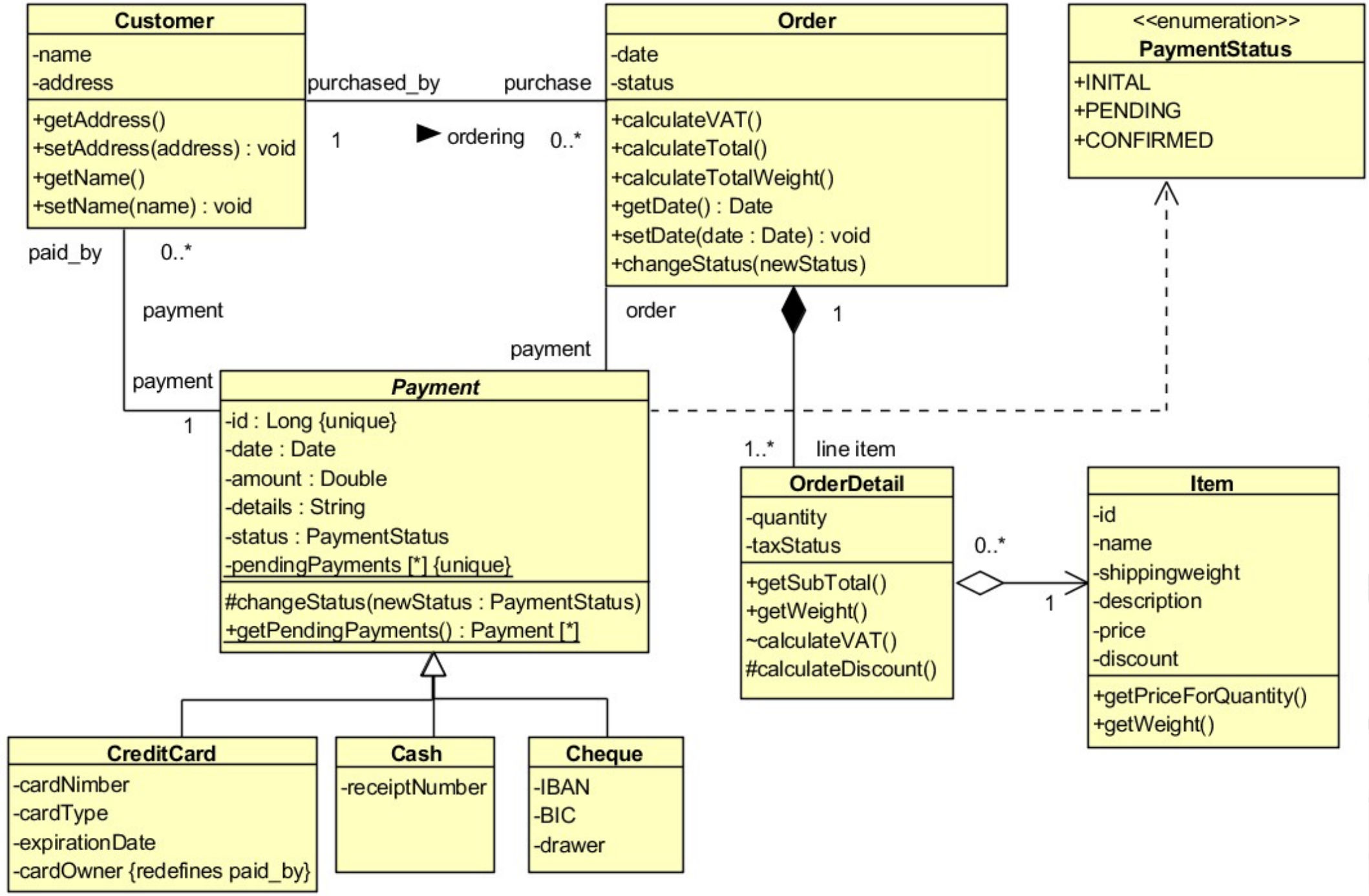
Обекти

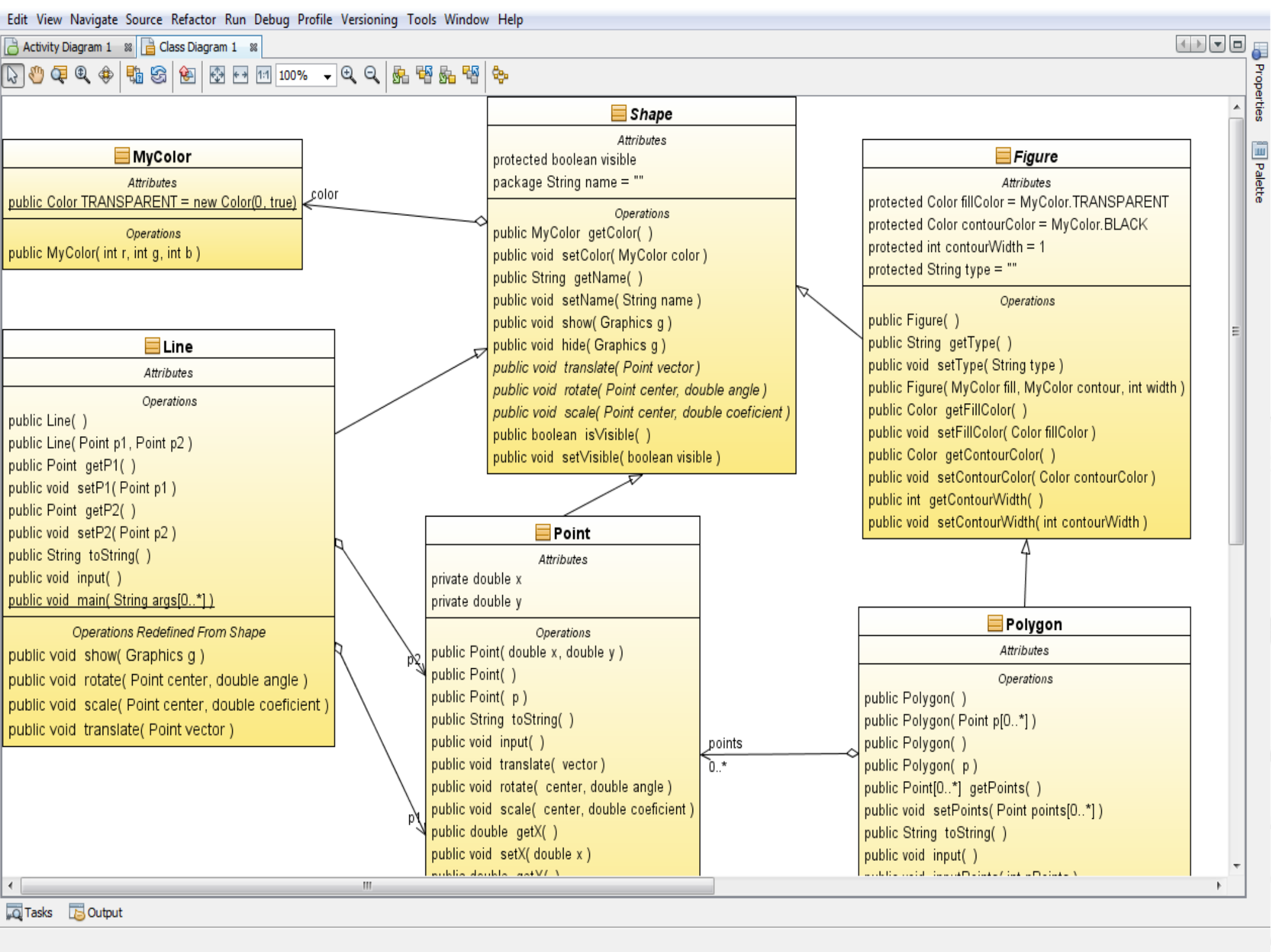
Спецификация на инстанция (Instance specification) = Обект – представлява инстанция на моделирана система, напр. клас --> обект, асоциация --> връзка (link), свойство --> атрибут и др

- може да осигури илюстрация или пример на обект
- описва обекта в конкретен момент от време
- може да бъде непълна
- Пример:

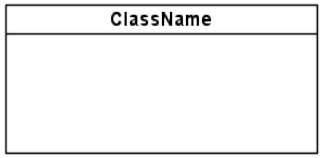


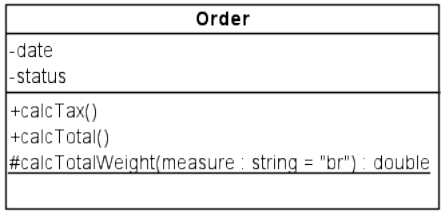
Класова диаграма (Class Diagram) - 1

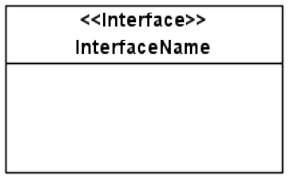


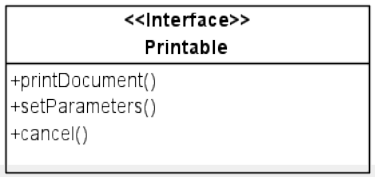

Edit View Navigate Source Refactor Run Debug Profile Versioning Tools Window Help
Activity Diagram 1
Class Diagram 1
100%
Properties
Palette
MyColor
Attributes
public Color TRANSPARENT = new Color(0, true)
Operations
public MyColor(int r, int g, int b)
color
Shape
Attributes
protected boolean visible
package String name = ""
Operations
public MyColor getColor( )
public void setColor(MyColor color)
public String getName( )
public void setName(String name)
public void show(Graphics g)
public void hide(Graphics g)
public void translate(Point vector)
public void rotate(Point center, double angle)
public void scale(Point center, double coeficient)
public boolean isVisible( )
public void setVisible(boolean visible)
Figure
Attributes
protected Color fillColor = MyColor.TRANSPARENT
protected Color contourColor = MyColor.BLACK
protected int contourWidth = 1
protected String type = ""
Operations
public Figure( )
public String getType( )
public void setType(String type)
public Figure(MyColor fill, MyColor contour, int width)
public Color getFillColor( )
public void setFillColor(Color fillColor)
public Color getContourColor( )
public void setContourColor(Color contourColor)
public int getContourWidth( )
public void setContourWidth(int contourWidth)
Line
Attributes
Operations
public Line( )
public Line(Point p1, Point p2)
public Point getP1( )
public void setP1(Point p1)
public Point getP2( )
public void setP2(Point p2)
public String toString( )
public void input( )
public void main(String args[0..*])
Operations Redefined From Shape
public void show(Graphics g)
public void rotate(Point center, double angle)
public void scale(Point center, double coeficient)
public void translate(Point vector)
p2
p1
Point
Attributes
private double x
private double y
Operations
public Point(double x, double y)
public Point( )
public Point(p)
public String toString( )
public void input( )
public void translate(vector)
public void rotate(center, double angle)
public void scale(center, double coeficient)
public double getX( )
public void setX(double x)
points
0..*
Polygon
Attributes
Operations
public Polygon( )
public Polygon(Point p[0..*])
public Polygon( )
public Polygon(p)
public Point[0..*] getPoints( )
public void setPoints(Point points[0..*])
public String toString( )
public void input( )
Tasks
Output

Елементи на класовите диаграми



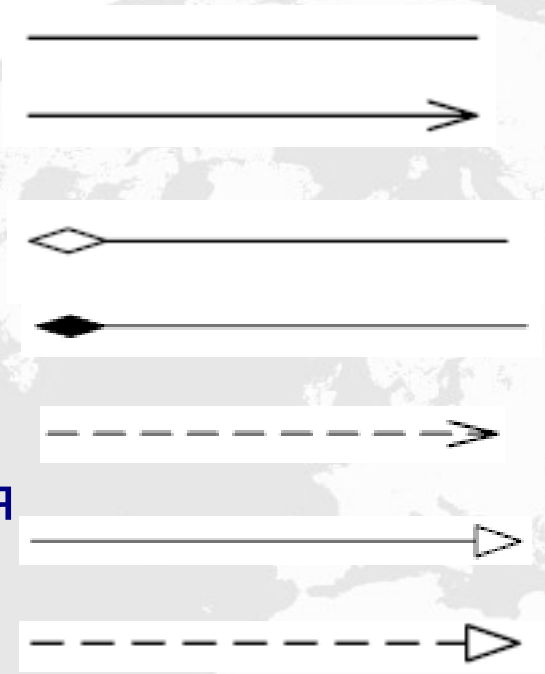






Видове връзки:

- асоциация
- агрегация
- композиция
- зависимост
- генерализация
- реализация



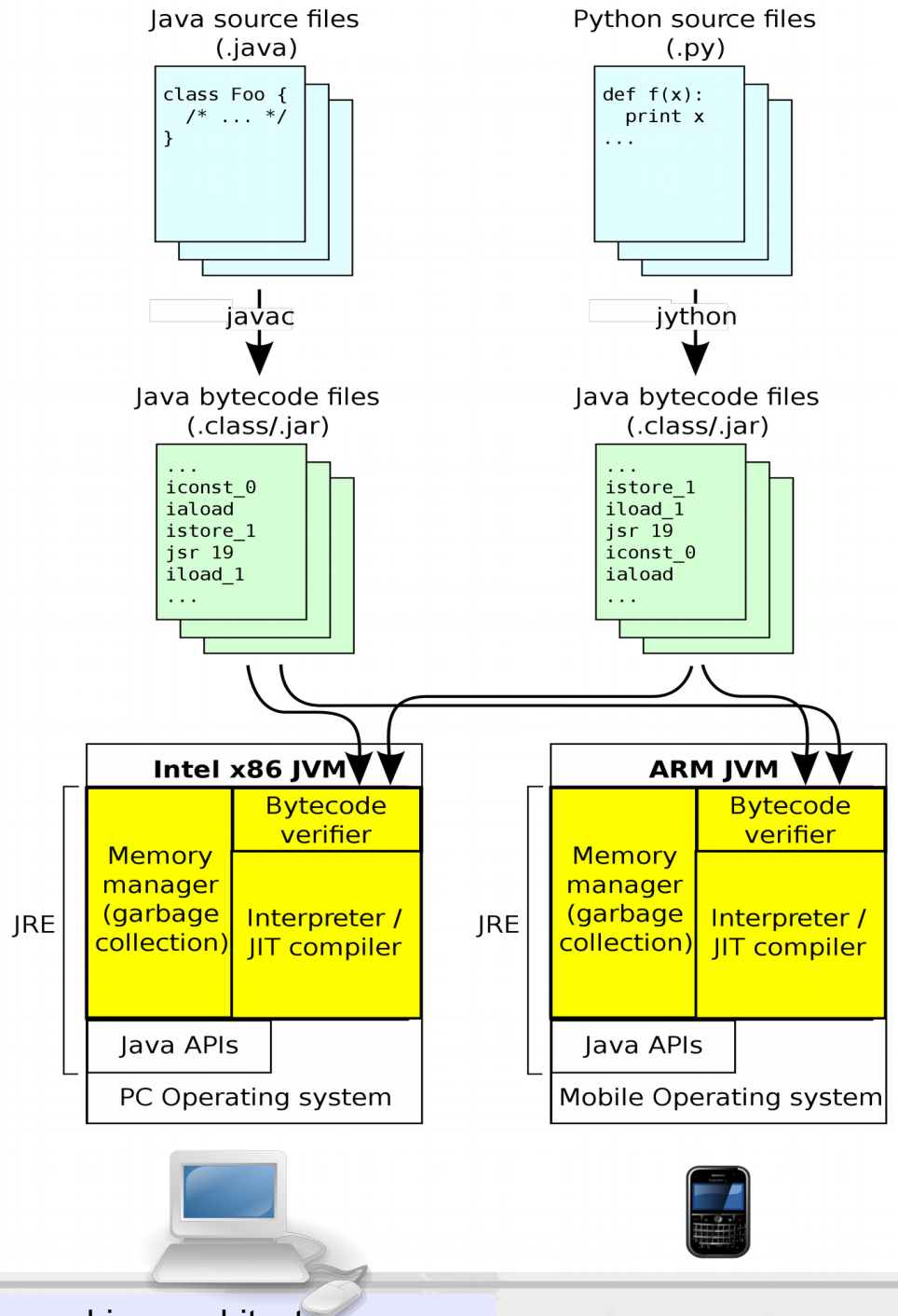
Основни характеристики на езика Java

- Еднобазова йерархия – наследяване само от един родителски клас, с възможност за имплементиране на множество интерфейси
- "Събирач на боклук" - Garbage Collector – преносимост и платформена независимост, по-малко грешки
- Сигурен код: разделяне на бизнес логиката от обработката на грешки и изключения
- Многонишковост – лесна реализация на паралелни обработки
- Персистентност – Java Database Connectivity (JDBC) и Java Persistence API (JPA)

Среда за разработка и изпълнение на Java™ приложения

- Версии на Java™ средата за разработка и изпълнение: JavaSE, JavaEE, JavaME, JavaFX
- Java Development Kit (JDK) и Java Runtime Environment (JRE)
- Java™ компилатор
- Виртуална машина
- Байт-код
- Инсталиране на JDK
- Компилиране и стартиране на програми от команден ред
- Среди за разработка: Eclipse, IntelliJ IDEA и NetBeans

Виртуална машина на Java™



Източник: http://en.wikipedia.org/wiki/File:Java_virtual_machine_architecture.svg, Автор: Dcoetzee, лиценз:Creative Commons CC0 1.0 Universal Public Domain Dedication

Методи за обектно-ориентирано проектиране на софтуер

- Итеративен и инкрементален процес
- Формулиране на потребителските изисквания
- Анализ и проектиране на архитектурата на системата
- Class- Responsibility-Collaboration (CRC) карти
- Гъвкави методологии (Extreme Programming – XP, Scrum)
- Open Unified Process (OpenUP) – подходящ за управление на малки проекти до 6 месеца
- Тестване на системата – unit тестове, интеграционно тестване, системно тестване, тестване на натоварване, ...

Социално кодиране с GitHub

- Системи за контрол на версиите и подпомагане на груповото писане на код: CVS, SVN, Git
- Система за контрол на версиите – позволява запазване на промените в кода по структуриран и управляем начин с възможност за възстановяване на предишно състояние на кода (rollback), експерименти (branches) и синхронизация на промените (merge)
- Това, което отличава Git е че промените се пазят локално под формата на „моментни снимки” (snapshots), вместо да се запазва списък на промените – бързи операции
- Три състояния: **Modified** → **Staged** → **Committed**

Основни команди на Git (1)

- Конфигуриране на Git

```
$ git config --global user.name "John Smith"
$ git config --global user.email jsmith@company.com
```

- Помощна информация за команда

```
$ git help <command_verb>
```

- Създаване на ново repository от съществуваща директория

```
$ git init
```

- Локално клониране на repository

```
$ git clone <repository_url> [<local_folder>]
```

Основни команди на Git (2)

- Добавяне на нови файлове – Staging и Commit

```
$ git add *.java
$ git add README.txt
$ git commit -m ”initial commit of MyProject”
```

- Информация за статуса на файловете в проекта

```
$ git status
```

- Показване на промените във файловете

```
$ git diff
```

- Игнориране на файлове – файл **.gitignore**

```
$ cat .gitignore
```

Основни команди на Git (3)

- Премахване на файлове

```
$ git rm README.txt
$ git commit -m ”removing README file from project”
```

- Преименуване на файлове

```
$ git mv README.txt README
```

- За повече информация:

http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository

- Примерни проекти за курса:

https://github.com/iproduct/course-java-npmg

Литература и интернет ресурси - I

- Booch, G., Object-Oriented Analysis and Design with Applications (2nd Edition), Addison-Wesley Professional; 2 edition, 1993
- Екел, Б., Да мислим на JAVA. Софтпрес, 2001.
- Oracle® Java™ Technologies webpage – http://www.oracle.com/technetwork/java/index.html
- Eclipse Application Development Framework – http://www.eclipse.org/
- Git Basics Tutorial – http://git-scm.com/book/en/v2/Getting-Started-Git-Basics
- Git How To – http://githowto.com/

Литература и интернет ресурси - II

- IPT проекти в GitHub хранилището – https://github.com/iproduct
- Object Management Group (OMG) UML™ webpage – http://www.uml.org/
- Фаулър, М., UML. Основи. Софтпрес, 2004
- Eriksson, H., Penker, M., UML Toolkit, Willey, 1998
- Wikipedia: Диаграма на класове – https://en.wikipedia.org/wiki/Class_diagram

Благодаря Ви за вниманието!

Въпроси?