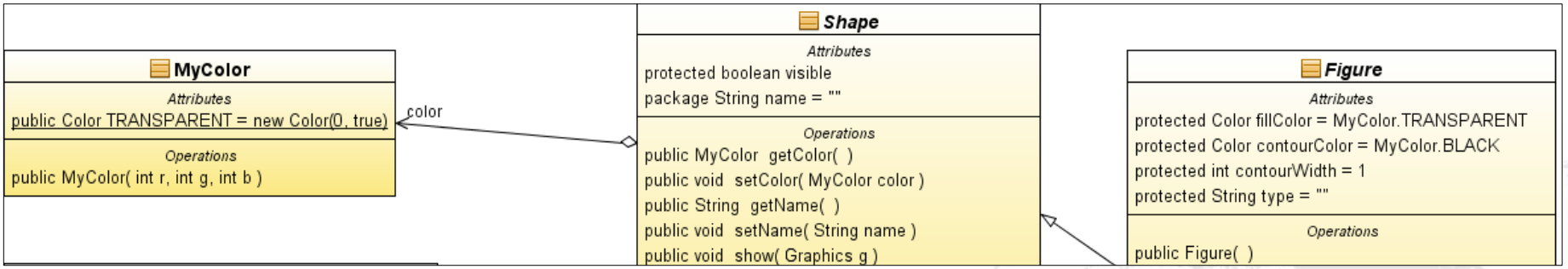
Trayan Iliev
IPT – Intellectual Products & Technologies Ltd.

Реализация на обектно-ориентирания подход за програмиране в езика Java™



Траян Илиев

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

Съдържание

1. Конструктори
2. Финализиране и garbage collector
3. Инициализация на членове на обекти
4. Инициализация на масиви
5. Пакети и спецификатори за достъп – public, private, protected
6. Многократно използване на класовете - композиция и наследяване
7. Презареждане и предефиниране на методи
8. Полиморфизъм
9. Абстрактни класове и методи
10. Интерфейси и вътрешни класове

Конструктори на обекти в езика Java™

- Инициализация на обектите с помощта на конструктори
- Презареждане (overloading) на конструктори и други методи
- Конструктори по подразбиране
- Референция към текущия обект – **this**

Възстановяване на паметта заета от ненужни обекти

- Има ли деструктори в езика Java™?
- "Чистач на боклук" - Garbage Collector
- Стратегии за освобождаване на заетата памет от ненужни обекти
  - stop-and-copy
  - mark-and-sweep
- Метод finalize()

Инициализация на членове на обекти. Инициализация на масиви

- Инициализация при деклариране
- Инициализация в конструктор
- Мързелива „lazy“ инициализация
- Инициализация на статични членове на клас
- Едномерни и многомерни масиви
- Инициализация на масиви

Пакети и спецификатори за достъп – public, private, protected

- Пакети и директории
- Импортиране на пакети – **import**
- Спецификатори за достъп
  - **public**
  - **private**
  - **protected**
- Приятелски достъп – по подразбиране, в рамките на пакета

Многократно използване на класовете

- Предимства от многократното използване на код
- Начини за реализация:
  - Композиция на обекти
  - Наследяване на класове (типове обекти)
- Реализация на композицията на езика Java™
  - Инициализиране на референциите:
    - при деклариране на обекта
    - в конструктора
    - преди използване (мързелива инициализация)

Наследяване на класове - 1

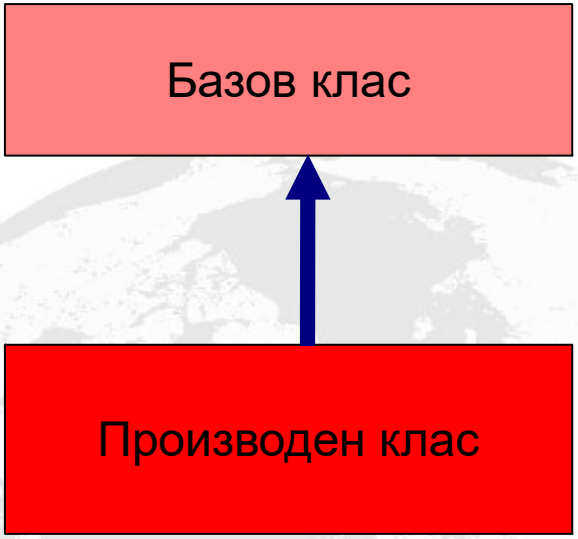
- Реализация на наследяването на езика Java™
  - Ключова дума **extends**
  - Ключова дума **super**
- Инициализация на обектите при наследяване: 1) базов клас; 2) наследен клас
  - Извикване на конструктори по подразбиране
  - Извикване на конструктори с аргументи
- Комбиниране на наследяване и композиция

Наследяване на класове - 2

- Изчистване на обектите - реализация в Java™
- Презареждане (overloading) и предефиниране (overriding) на методи на базовия клас в производни класове
- Кога да използваме композиция и кога наследяване?
  - Нуждаем ли се от интерфейса на базовия клас?
  - Тип на връзката - **„има“** и **„е“**?

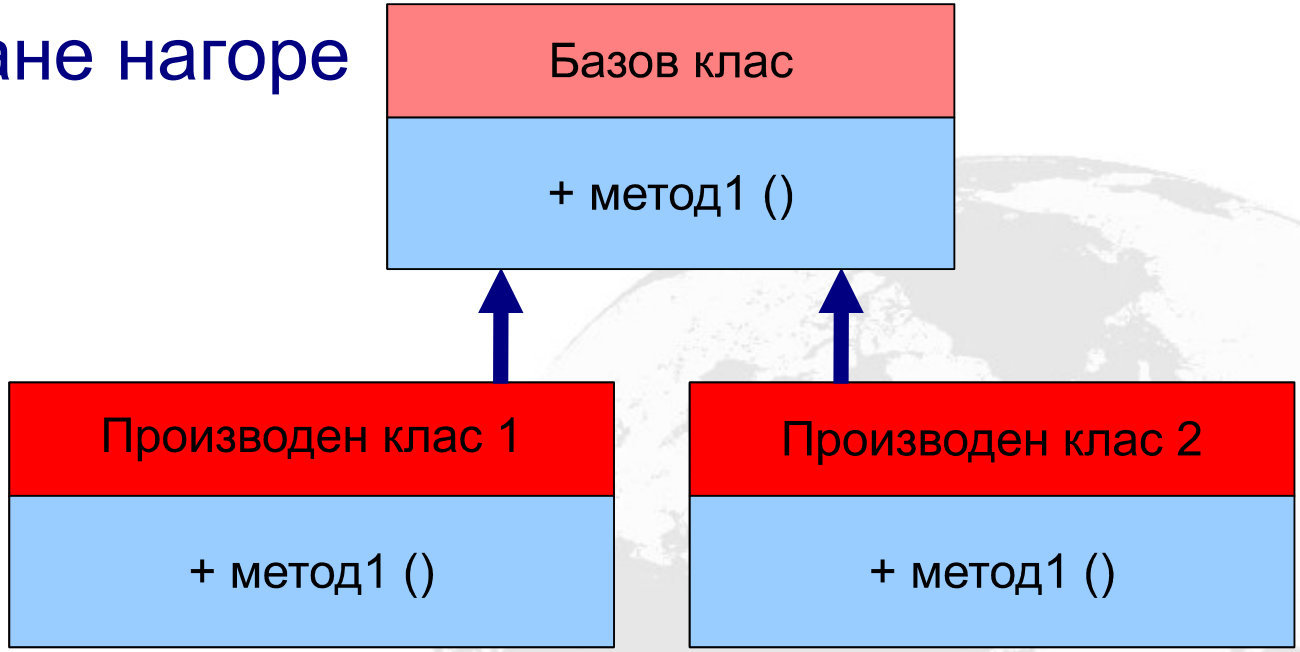
Наследяване на класове - 3

- **Protected** методи
- Преобразуване нагоре
- Ключова дума **final**
  - Final данни – дефиниране на константи
    - данни от прост тип
    - обекти
    - празни полета
    - аргументи
  - Final методи
  - Final класове



Полиморфизъм (1)

- Преобразуване нагоре



- Абстрактни методи и класове – **abstract**
- Ред на извикване на конструкторите
- Наследяване и разширяване

Полиморфизъм (2)

- Полиморфизъм – по-подразбиране, освен ако методът е деклариран като **static** или **final** (**private** методите стават автоматично **final**)
- При **конструиране на обектите с наследяване** всеки обект се грижи за своите атрибути и делегира инициализацията на родителските атрибути на родителския конструктор или метод
- Използване на полиморфни методи в конструктор
- **Ковариантни** типове на връщане (от Java SE 5)
- Композиция <-> Наследяване -**State** Design Pattern

Интерфейси и множествено наследяване

- Интерфейси – ключови думи: **interface**, **implements**
- Множествено наследяване в Java™
- Разширяване на интерфейс чрез наследяване
- Константи (**static final**)
- Влагане на интерфейси

Предимства от използването на интерфейси

- **Интерфейсите** чисто разделят изисквания тип на обекта от множеството възможни реализации и правят нашия код по-универсален и използваем
- Reusable Design Pattern: **Adapter** – позволява да адаптираме вече съществуваща реализация към интерфейса, който се изисква в нашето приложение
- Наследяване (разширяване) на интерфейси
- Reusable Design Pattern: **Factory Method** – създаваме многократно използваем клиентски код, изолиран от спецификата на конкретната сървърна реализация

Вътрешни класове (1)

- **Вътрешните класове** групират логически свързаните класове и контролират тяхната видимост
- **Closures** – вътрешния клас има постоянна връзка към съдържащия го външен клас и може да достъпва всичките му атрибути и дори **final** аргументи и локални променливи (ако е дефиниран в метод или блок)
- Вътрешните класове могат да са **анонимни**, ако се ползват еднократно в програмата. Конструиране.
- Референция към обекта от външен клас - **.this** и създаване на обект от вътрешния клас в **контекста** на съдържащия го обект от външния клас - **.new**

Вътрешни класове (2)

- Вътрешни класове
  - дефинирани във външен клас
  - дефинирани в метод
  - дефинирани в блок от оператори
  - достъп до атрибутите на външния клас и до аргументите на метода в който са дефинирани
- Анонимни вътрешни класове
  - реализиращи публичен интерфейс
  - наследяващи клас
  - инициализация на инстанция
  - статични вътрешни класове

Литература и интернет ресурси

- Екел, Б., Да мислим на JAVA. Софтпрес, 2001.
- Eckel, B., Thinking in Java, 4th edition, Prentice Hall, 2006, http://mindview.net/Books/TIJ4
- Oracle® Java™ Technologies webpage – http://www.oracle.com/technetwork/java/index.html

Благодаря Ви за вниманието!

Въпроси?