Java SE
5. ООП в Java

© 2011 NetCracker Technology Corp. Confidential.

Основные понятия

© 2011 NetCracker Technology Corp. Confidential.

Классы

- Классы объявляют новый ссылочный тип и определяют его реализацию
- Вложенный (nested) класс – это класс объявленный внутри другого класса или интерфейса (в том числе класс объявленный внутри метода или блока):
  - member class – объявленный внутри класса
  - local class – объявленный внутри метода
  - anonymous class – не имеющий имени
- Класс верхнего уровня(top-level) – это класс, не являющийся вложенным

© 2011 NetCracker Technology Corp. Confidential.

Классы

- Каждый класс кроме Object является наследником другого класса и может реализовывать (implements) произвольное количество интерфейсов
- Тело класса может содержать:
  - члены (members):
    - поля
    - методы
    - вложенные классы и интерфейсы
  - инициализаторы экземпляра
  - статические инициализаторы
  - конструкторы
- Видимость членов и конструкторов регулируется модификаторами доступа

  public, private, package(default), protected

© 2011 NetCracker Technology Corp. Confidential.

Пример объявления класса

```
abstract class MyClass extends Parent
        implements MyInterface, AnotherInterface
{
    static {
        //Static initializer
    }
    {
        //Non static initializer
    }
    public MyClass()
    {
        super(); //Вызов конструктора суперкласса
    }
}
```

- Первым будет выполнен статический инициализатор (при загрузке класса в JVM)
- Не статические инициализаторы будут выполнены при создании объекта в порядке объявления

© 2011 NetCracker Technology Corp. Confidential.

Классы и наследование

- Java не поддерживает множественного наследования классов
- Членами класса являются унаследованные и определенные в классе члены
- Вновь объявленные поля могут скрывать поля суперклассов и суперинтерфейсов
- Вновь объявленные методы могут скрывать, реализовывать или перегружать методы, объявленные в суперклассе или суперинтерфейсе
- Вложенные классы бывают статическими и внутренними (inner), в зависимости от контекста в котором они объявлены (если в точке объявления имеет смысл ссылка this – то вложенный класс будет внутренним)

© 2011 NetCracker Technology Corp. Confidential.

Интерфейсы

- Определяют контракт, не зависящий от конкретной реализации
- В объявлении используется ключевое слово **interface**
- Класс может реализовывать до 65 535 интерфейсов
- Реализацией нескольких интерфейсов можно эмулировать множественное наследования контрактов (поведения), но не состояния
- Все методы интерфейса **public**
- Все поля – **public static final**
- Интерфейс может наследовать другие интерфейсы при помощи ключевого слова **extends**

```
interface A {
    int getValue();
}

interface C {
    int getValue();
}

public class Correct implements A, C {
    public int getValue() {
        return 5;
    }
}
```

© 2011 NetCracker Technology Corp. Confidential.

Интерфейсы (Java 8)

- Могут определять методы по умолчанию (default interface methods)

```
interface Formula {
    double calculate(int a);
    default double sqrt(int a) {
        return Math.sqrt(a); }
}
```

Классы, имплементирующие этот интерфейс, должны переопределить только абстрактный метод calculate. Метод по умолчанию sqrt будет доступен без переопределения.

```
Formula formula = new Formula() {
    @Override public double calculate(int a) {
        return sqrt(a * 100); }
};
formula.calculate(100); // 100.0
formula.sqrt(16); // 4.0
```

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Абстрактный класс

- Класс **C** является абстрактным, если выполняется хотя бы одно из следующих условий:
  - Класс **C** явно содержит объявление абстрактного метода
  - Какой-либо класс-родитель **C** содержит объявление абстрактного метода, который не был реализован в классе **C** или в его родительских классах
  - Один из интерфейсов **C** определяет или наследует метод, который не реализован (и поэтому является абстрактным), т. е. ни **C** ни его родительские классы не определяют реализацию этого метода
  - Класс **C** объявлен с модификатором **abstract**
- Экземпляр абстрактного класса создать нельзя
- Суть абстрактного класса в частичной реализации требуемого поведения. Остальное предстоит реализовать подклассам
- Иногда классы делают абстрактными чтобы запретить создание экземпляров.

© 2011 NetCracker Technology Corp. Confidential.

Классы и полиморфизм

- Полиморфизм – вариация поведения в зависимости от конкретной реализации в рамках единого контракта
- Все методы в Java потенциально являются виртуальными
- Тем не менее, полиморфный вызов будет происходить не всегда
  - **Private**-методы не полиморфны
  - **Static**-методы не полиморфны
  - **Final**-методы не полиморфны
- Хотя неполиморфный вызов в теории производительнее полиморфного, не нужно пытаться подсказывать JVM, что метод не полиморфен при помощи модификаторов
- Полиморфизм позволяет абстрагироваться от деталей реализации и работать с объектом как с «черным ящиком»

NetCracker® Accelerating Business Transformation™
© 2011 NetCracker Technology Corp. Confidential.

Перегрузка методов (Overriding)

- Перегрузка методов экземпляра позволяет создать условия для полиморфного вызова
- При перегрузке метода можно изменять объявление метода в пределах, не нарушающих исходного контракта
  - Возвращаемое значение может стать более конкретным
  - Тип параметра может стать более общим
  - Объявленное в заголовке исключение может быть опущено
  - Область видимости может быть расширена
- Общее правило – перегруженный метод можно использовать вместо исходной версии без ошибок компиляции
- Аннотация **@Override** позволяет на этапе компиляции проверить правильность перегрузки методов

NetCracker® Accelerating Business Transformation™
© 2011 NetCracker Technology Corp. Confidential.

Полиморфизм: пример

```
public class Parent {
    public int getValue() {
        return 5;
    }
}

class Child extends Parent {
    @Override
    public int getValue() {
        return 10;
    }

    public static void main(String[] args) {
        Parent parent = new Child();
        System.out.println(parent.getValue()); // 10
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Модификаторы объявления класса

- **public** – класс доступен извне пакета.
- **abstract** – класс является абстрактным (в нем есть абстрактные методы)
- **final** – класс является конечным в иерархии наследования. От него нельзя унаследовать другой класс
- **strictfp** – для всех методов класса действуют правила строгой проверки арифметических выражений во время вычислений

Для вложенных(внутренних) классов дополнительно действуют следующие модификаторы:

- **static –** класс является статическим (вложенный класс)
- **protected** – к классу имеют доступ только классы наследники объемлющего класса или классы в том же пакете
- **private** – к классу имеет доступ только объемлющий класс

Модификаторы static и final

© 2011 NetCracker Technology Corp. Confidential.

Final-поля

- Переменные классов (`static`) и переменные экземпляров могут быть объявлены `final`
- Статическая переменная, объявленная `final` должна быть инициализирована непосредственно при объявлении либо в блоке статической инициализации
- Переменная экземпляра (не `static`), объявленная `final` должна быть проинициализирована непосредственно при объявлении либо в блоке инициализации, либо ей должно быть присвоено значение к концу исполнения *каждого* конструктора
- Final-переменная после инициализации не может изменить своего значения. Единственный способ изменить его искусственно – использовать Reflection API

© 2011 NetCracker Technology Corp. Confidential.

Final-поля: пример

- Переменная ,объявленная **final**, может быть проинициализирована ровно один раз, после этого ее значение изменить нельзя.

```
void flow(boolean flag) {
    final int k;
    if (flag) {
        k = 3;
    } else {
        k = 4;
    }
    System.out.println(k);
    // OK
}
```

```
void flow(boolean flag) {
    final int k;
    if (flag) {
        k = 3;
    }
    if (!flag){
        k = 4;
    }
    System.out.println(k);
} //compile-time error!
```


© 2011 NetCracker Technology Corp. Confidential.

Static

- Статические поля и методы являются атрибутами класса, а не объекта
- Все экземпляры класса (объекты) будут работать с одним и тем же статическим полем
- Комбинация модификаторов **static final** по сути означает объявление константы
- В статических методах ссылки **this** и **super** будут недоступны – в контексте нет объекта
- Из **static**-методов можно оперировать только **static**-полями
- Обращение к статическим членам класса происходит через точку с указанием класса, а не конкретного объекта.
- Последний вариант, впрочем, технически возможен, но считается плохим стилем программирования

© 2011 NetCracker Technology Corp. Confidential.

Пример static-поля

```
class Point {
    int x, y, useCount;
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    final static Point origin = new Point(0, 0);
}

class Test {
    public static void main(String[] args) {
        Point p = new Point(1,1);
        Point q = new Point(2,2);
        p.x = 3;
        p.y = 3;
        p.useCount++;
        p.origin.useCount++;
        System.out.println(q.useCount);              //0
        System.out.println(q.origin == Point.origin); //true
        System.out.println(q.origin.useCount);        //1
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Члены классов

Конструкторы и инициализаторы

- Инициализаторы экземпляра – блоки кода **{…}** выполняемые при инициализации объекта. Выполняются перед вызовом конструктора.
- Статические инициализаторы – статические блоки кода **static {…}** выполняемые при загрузке класса JVM перед первым использованием
- Конструкторы, в отличие от методов, не могут быть вызваны непосредственно с помощью вызова через точку. Конструкторы вызываются при создании объектов и могут быть перегружены

© 2011 NetCracker Technology Corp. Confidential.

Методы

- Методы описывают участки кода, которые могут быть вызваны с помощью выражения вызова метода
- Метод класса исполняется в контексте переменных класса (static context)
- Метод экземпляра исполняется в контексте конкретного объекта, доступного по `this`
- Методы не имеющие реализации должны быть объявлены abstract
- Допускается перегрузка методов по списку и типам аргументов
- Метод может иметь платформенно-зависимую реализацию (native method). В таких методах недоступен debug и просмотр исходного кода

© 2011 NetCracker Technology Corp. Confidential.

Параметры метода

- Во время вызова метода вычисленные значения передаваемых аргументов используются для инициализации переменных-параметров метода
- Таким образом всегда имеет место передача «по значению»
- Область видимости параметра ограничивается методом, в котором он объявлен. При этом доступ к нему осуществляется с помощью обычного имени.
- Параметры перекрывают собой поля-члены класса, в котором объявлен метод, содержащий эти параметры. Для доступа к перекрытым полям-членам нужно использовать **this** либо полное квалифицированное имя
- Параметры типов double и float всегда содержат значения из множества double и float соответственно. Они не могут принимать расширенных значений появляющихся во время вычисления выражений не являющихся **strictfp**

Поля в классах

- Поля в классах могут иметь либо примитивный, либо ссылочный тип
- Область видимости полей, в том числе по цепочке наследования, регулируется модификаторами доступа
- При этом поля класса скрывают одноименные поля суперклассов. Для обращения к скрытым полям можно использовать нотацию вида **super.fieldName;**
- При создании объекта (загрузке класса в случае статических полей) будет выполнена инициализация полей значениями по умолчанию.
- В многопоточном окружении, тем не менее, возможна ситуация, в которой сторонний поток может получить не полностью сконструированный объект

© 2011 NetCracker Technology Corp. Confidential.

Поля в интерфейсах

- Фактически все поля интерфейса являются **public static final** константами. Декларация данных спецификаторов является избыточной.
- Каждое поле в теле интерфейса должно быть проинициализировано выражением, значение которого должно быть вычислено на стадии компиляции. При этом возможно использование в выражении уже проинициализированных полей самого интерфейса
- Поля инициализируются в порядке их декларации за исключением полей, явно инициализируемых константами.
- В выражениях инициализации нельзя использовать ключевые слова **this** и **super** кроме случая если эти слова используются внутри декларации тела анонимного класса реализующего интерфейс.

```
interface Test {
    float f = j; //error - j используется до объявления
    int j = 1;
    int k = k+1; //error - k инициализируется с использованием k
}
```

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential. *

Внутренние классы

© 2011 NetCracker Technology Corp. Confidential.

Внутренние (inner) классы

- Внутренний (**inner**) класс – это класс, определенный в контексте другого класса
- Внутренние классы делятся на статические и нестатические
- Статические вложенные классы, не имеют доступа к нестатическим полям и методам обрамляющего класса, так как статика соотносится к классом и не имеет отношения к любому из конкретных объектов
- Объект нестатический внутреннего класс может быть создан только в контексте объекта внешнего класса и имеет одинаковый с ним жизненный цикл

```
public class ClassToTest {
    private static void internalMethod() {  }

    public static class Test {
        public void testMethod() {  }
    }
}
```

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Нестатические внутренние классы

- Применяются, в частности, для моделирования отношения композиции
- Могут быть нескольких видов
  - внутренние классы-члены (member inner classes). Объявляются в контексте класса
  - локальные классы (local classes). Объявляются внутри методов
  - анонимные классы (anonymous classes). Не имеют явного имени и, как правило, у такого класса создается единственный объект прямо в месте объявления
- В результате компиляции из каждого из них будет создан отдельный .class-файл с префиксом из имени внешнего класса.

NetCracker® Accelerating Business Transformation™ © 2011 NetCracker Technology Corp. Confidential.

Пример нестатического внутреннего класса

```
class WithDeepNesting {
    boolean toBe;

    WithDeepNesting(boolean b) {
        toBe = b;
    }

    class Nested {

        boolean theQuestion;

        class DeeplyNested {
            DeeplyNested(){
                theQuestion = toBe || !toBe;
            }
        }
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Пример локального внутреннего класса

```java
public class Handler {
    public void handle(String requestPath) {
        class Path {

            String path;

            Path(String path) {
                this.path = path;
            }

            boolean startsWith(String s) {
                return path.startsWith(s);
            }
        }

        Path path = new Path(requestPath);

        if (path.startsWith("/page")) {

        }
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Пример анонимной реализации интерфейса

- Аналогичным образом можно оформлять анонимное наследование от конкретных классов

```
public interface Interface {
    void doIt();
}

class Class {

    public static void main(String[] args) {
        Interface impl = new Interface() {
            public void doIt() {
                System.out.println("Hello!");
            }
        };
        impl.doIt();
    }
}
```

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Инициализация

© 2011 NetCracker Technology Corp. Confidential.

Инициализация полей

- Инициализация полей экземпляра происходит каждый раз при создании нового объекта.
- Инициализация статических полей класса происходит один раз при первом использовании класса.
- При инициализации поля экземпляра могут использовать статические поля, т.к. они гарантированно инициализированы к моменту создания объекта
- При инициализации статические поля класса не могут использовать поля экземпляра, а также ключевые слова `this` и `super`
- Инициализация полей происходит в порядке объявления и в порядке исполнения блоков инициализации. Код конструкторов исполняется в последнюю очередь

NetCracker® Accelerating Business Transformation™ © 2011 NetCracker Technology Corp. Confidential.

Overloading, Overriding и Hiding

```
class Point {
    int x = 0, y = 0;
    int color;
    void move(int dx, int dy) {
        x += dx;
        y += dy;
    }
}
class RealPoint extends Point {
    float x = 0.0f, y = 0.0f; // hiding x and y
    // overriding move
    void move(int dx, int dy) {
        move((float)dx, (float)dy);
    }
    //overloading move
    void move(float dx, float dy) {
        x += dx;
        y += dy;
    }
}
```


© 2011 NetCracker Technology Corp. Confidential.

Перечислимые типы

© 2011 NetCracker Technology Corp. Confidential.

Перечисления (enum)

- В отличие от статических констант, предоставляют типизированный, безопасный способ задания фиксированных наборов значений
- Являются классами специального вида, не могут иметь наследников, сами в свою очередь наследуются от **java.lang.Enum**. Не могут быть абстрактными и содержать абстрактные методы, но могут реализовывать интерфейсы
- Экземпляры объектов перечисления нельзя создать с помощью new, каждый объект перечисления уникален, создается при загрузке перечисления в виртуальную машину, поэтому допустимо сравнение ссылок для объектов перечислений, можно использовать **switch**
- Как и обычные классы могут реализовывать поведение, содержать вложенные и внутренние классы-члены

© 2011 NetCracker Technology Corp. Confidential.

Пример

```java
public enum Days {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY;

    public boolean isWeekend() {
        switch(this) {
            case SUNDAY:
            case SATURDAY:
                return true;
            default:
                return false;
        }
    }
}

  Days day = Days.MONDAY;
  Days anotherDay = Days.valueOf("tuesday");
  day.isWeekend(); // false
```

NetCracker® Accelerating Business Transformation™ © 2011 NetCracker Technology Corp. Confidential.

Enum API

- Каждый класс перечисления неявно содержит следующие методы:
  - **`values()`** - возвращает массив элементов перечисления (статический метод)
  - **`ordinal()`** - возвращает порядковый номер элемента перечисления (в порядке декларации)
  - **`valueOf(String name)`** – возвращает элемент перечисления по его строковому имени (статический метод, выбрасывает **`IllegalArgumentException`** если нет элемента с указанным именем)
- Класс перечисления может иметь конструктор (private либо package), который вызывается для каждого элемента при его декларации
- Отдельные элементы перечисления могут реализовывать свое собственное поведение

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Пример более сложного enum

```java
public enum Direction {
    FORWARD(1.0) {
        public Direction opposite() {
            return BACKWARD;
        }
    },
    BACKWARD(2.0) {
        public Direction opposite() {
            return FORWARD;
        }
    };

    Direction(double r) {
        // some business logic
    }

    public abstract Direction opposite();
}
```


© 2011 NetCracker Technology Corp. Confidential.

Пакеты и import

© 2011 NetCracker Technology Corp. Confidential.

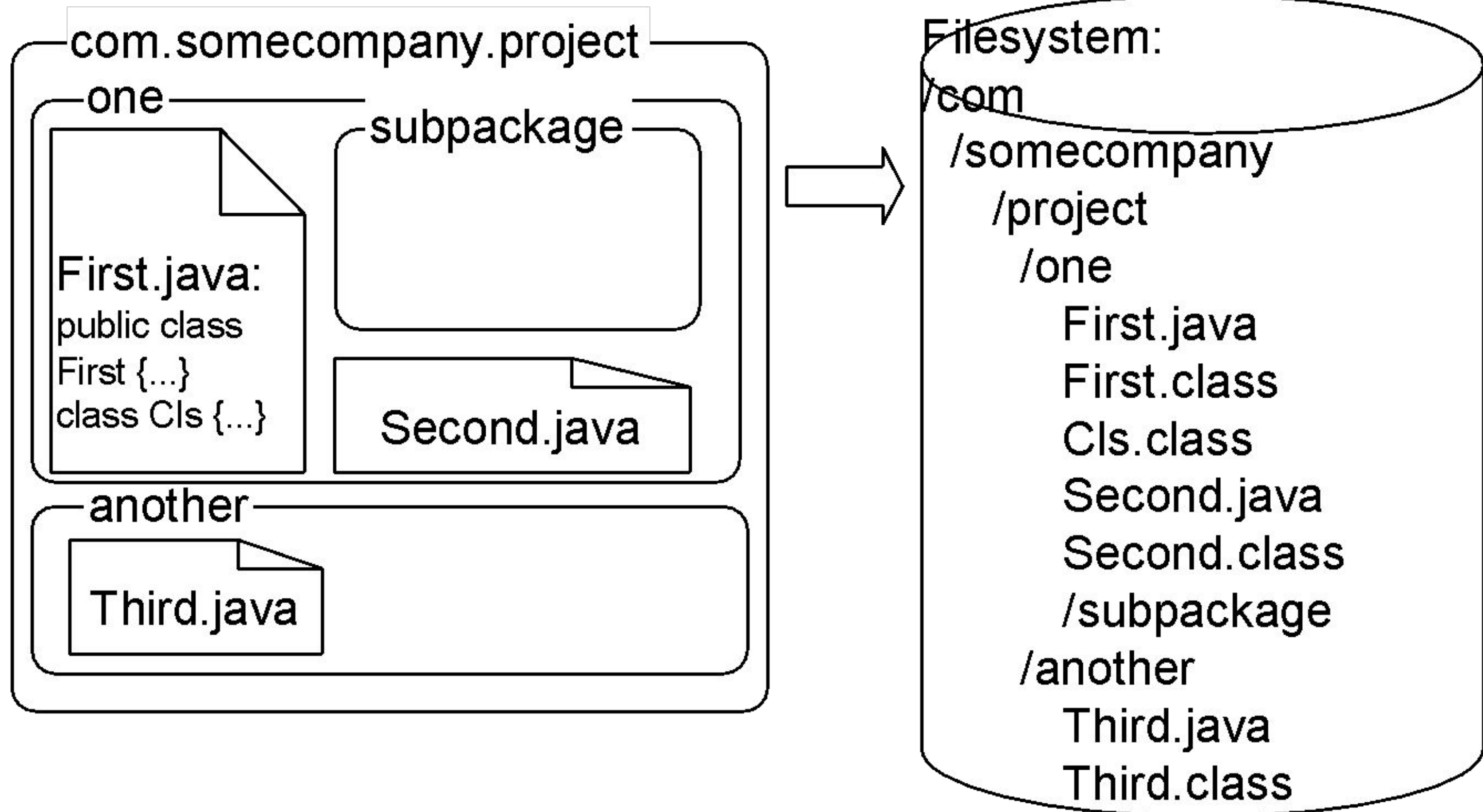
Пакеты

- В типичном случае программа состоит из нескольких пакетов
- Каждый пакет имеет собственное пространство имен для типов объявленных в пакете
- Тип доступен извне пакета только если он объявлен с модификатором **public**
- Пакеты образуют иерархическую структуру имен.
- Членами пакета являются:
  - Классы, интерфейсы, enum, объявленные в пакете
  - подпакеты, которые имеют свои собственные подпакеты и классы
- Пакеты имеют ряд ограничений на их организацию для обеспечения однозначности при поиске и загрузке (а также компиляции) типов

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Пакеты: файловая структура



- При отображении (хранении) на файловой системе один файл может содержать только один тип объявленный public совпадающий по имени с именем файла

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential. *

Декларация import

- Для того чтобы получить доступ к членам другого пакета (кроме java.lang) в классе, нужно явно импортировать эти члены. По умолчанию они не видны

```
import java.io.InputStream; // single type import declaration
import java.net.*;          // import on demand declaration
```

- Будучи импортированными типы становятся доступны в единице компиляции с использованием простого (не квалифицированного) имени.
- Вопреки распространенному мнению лишние импорты не приводят к автоматической загрузке классов или любой другой напрасной трате ресурсов. JVM загружает класс по факту использования, а не импортирования

© 2011 NetCracker Technology Corp. Confidential.

Статический импорт

- Для того чтобы иметь возможность обращаться к статическим методам, полям класса, а также к элементам перечислений без использования квалифицированного имени, можно воспользоваться статической декларацией импорта:

```
import static Days.* ;
import static java.lang.Math.*;
…
Day d = MONDAY;
Day d2 = valueOf("SATURDAY");
double v = sin(PI/2);
```

- Однако злоупотреблять статическим импортом не стоит

NetCracker® Accelerating Business Transformation™

© 2011 NetCracker Technology Corp. Confidential.

Правила именования пакетов

- Для обеспечения уникальности имени пакета в качестве основы следует использовать доменное имя организации, например: **ru.nsu.fit.mylastname.task1**
- Примеры имен:

```
com.sun.java.jag.scrabble
com.apple.quicktime.v2
com.novosoft.siberon.someproject
```


© 2011 NetCracker Technology Corp. Confidential.

Q&A

© 2011 NetCracker Technology Corp. Confidential.

© 2011 NetCracker Technology Corp. Confidential.