Урок 5.
Проектирование классов (продолжение)

Гарбузов Николай
Donriver Inc.

DonRiver

Agenda

- Абстрактные классы
- Наследование, порядок инициализации класса
- Ключевое слово super
- Переопределение методов
- Интерфейсы
- Анонимные классы

Абстрактные классы

Абстрактный класс задает заданную абстракцию, не предоставляя реализации каждого метода

- его нельзя создать оператором new
- используют ключевое слово abstract к методам и классам

Абстрактные классы

Абстрактный класс задает заданную абстракцию, не предоставляя реализации каждого метода

- его нельзя создать оператором new
- используют ключевое слово abstract к методам и классам

```
abstract class Employee {
        protected String name;

        public abstract void work(int hours);
}
```

Наследование

Q:
Как унаследоваться от класса?

Наследование

Q:
Как унаследоваться от класса?

A:
Использовать extends

```
class Engineer extends Employee {
        @Override
        public void work(int hours){
                //do work here
        }
}
```

Наследование

Q:
Как унаследоваться от класса?

A:
Использовать extends

```
class Engineer extends Employee {
        @Override
        public void work(int hours){
                //do work here
        }
}
```

Наследование

Унаследованные классы могут:

- использовать унаследованные protected, public методы и поля
- перегружать методы
- переопределять методы
- вызывать конструктор родительского класса

Наследование

Q:
Когда наследовать класс B от класса A?

Наследование

Q:
Когда наследовать класс B от класса A?

A:
Если класс B находится в отношении “является” (“is-a”) к классу A

Наследование

Q:
Когда наследовать класс B от класса A?

A:
Если класс B находится в отношении “является” (“is-a”) к классу A

Ex:

- Прямоугольник “is-a” фигура
- Программист “is-a” рабочий
- NTFS “is-a” файловая система

Ключевое слово super

Используется для доступа из унаследованного класса к _____ родительского класса:

- полю
- конструктору
- методу

Переопределение методов

Метод A переопределяет метод B, если:

- A находится в потомке класса, которому принадлежит B
- они имеют одинаковую сигнатуру

Переопределение методов

Метод A переопределяет метод B, если:

- A находится в потомке класса, которому принадлежит B
- они имеют одинаковую сигнатуру

- возвращаемый тип метода A ковариантен с возвращаемым типом метода B

Переопределение методов

Метод A переопределяет метод B, если:

- A находится в потомке класса, которому принадлежит B
- они имеют одинаковую сигнатуру

- возвращаемый тип метода A ковариантен с возвращаемым типом метода B

- список исключений, объявленных в методе A, является подмножеством исключений метода B

Переопределение методов

Q:
В чем же смысл переопределения?

Переопределение методов

Q:
В чем же смысл переопределения?

A:
В **динамическом полиморфизме**!

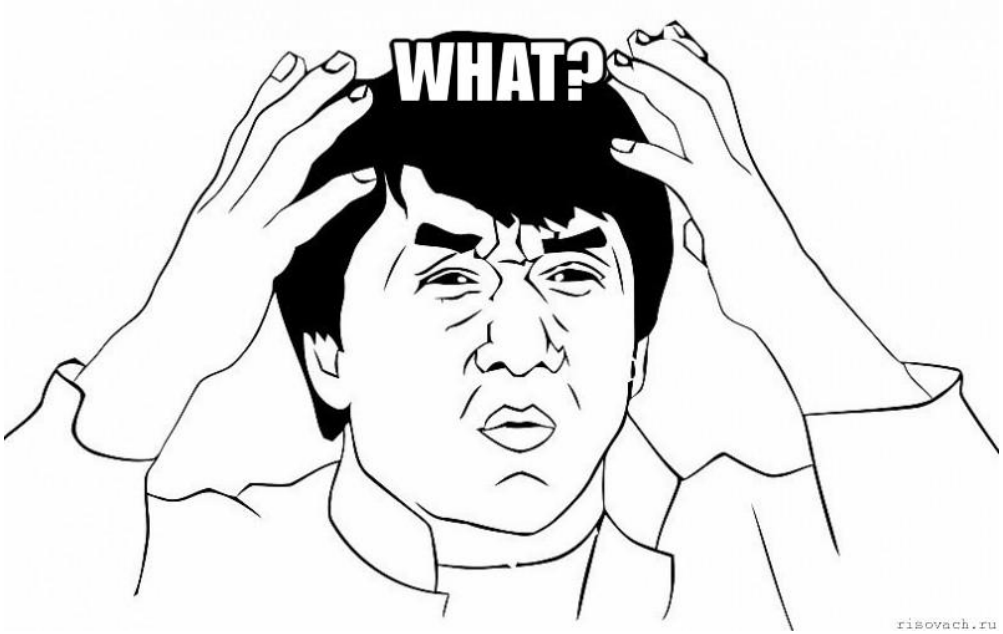
достигается с помощью динамической диспетчеризации - механизма определения вызова переопределенного метода не во время компиляции, а во время выполнения!

Интерфейсы

Интерфейсы определяют “контракт” класса и предназначены для “динамического разрешения вызовов методов во время выполнения”

Интерфейсы

Интерфейсы определяют “контракт” класса и предназначены для “динамического разрешения вызовов методов во время выполнения”



DonRiver

Интерфейсы

- Интерфейсы указывают “что делать”, а не “как делать”
- Различные реализации интерфейса A могут быть использованы как значения переменной с типом интерфейса A

Интерфейсы

- Интерфейсы реализуются (имплементируются) с помощью ключевого слова implements
- Класс обязан реализовывать все методы либо являться абстрактным

Интерфейсы

- Интерфейсы реализуются (имплементируются) с помощью ключевого слова implements
- Класс обязан реализовывать все методы либо являться абстрактным (до Java 8)

```
abstract interface Hero {
    public abstract Good feat (Evil e);
}
```

```
class Superman implements Hero {
    public SuperGood hit (Evel e) {
        // save earth
        return new SuperGood(...);
    }
}
```

```
class abstract MarvelHero
        implements Hero {}
```

Интерфейсы

- Интерфейсы реализуются (имплементируются) с помощью ключевого слова implements
- Класс обязан реализовывать все методы либо являться абстрактным (до Java 8)

```
abstract interface Hero {
    public abstract Good feat (Evil e);
}
```

```
class Superman implements Hero {
    public SuperGood hit (Evel e) {
        // save earth
        return new SuperGood(...);
    }
}
```

```
class abstract MarvelHero
    implements Hero {}
```

Интерфейсы

- Интерфейсы реализуются (имплементируются) с помощью ключевого слова implements
- Класс обязан реализовывать все методы либо являться абстрактным (до Java 8)

```
abstract interface Hero {
    public abstract Good feat (Evil e);
}
```

```
class Superman implements Hero {
    public SuperGood hit (Evel e) {
        // save earth
        return new SuperGood(...);
    }
}
```

```
class abstract MarvelHero
        implements Hero {}
```

Интерфейсы

- Интерфейсы реализуются (имплементируются) с помощью ключевого слова implements
- Класс обязан реализовывать все методы либо являться абстрактным (до Java 8)

```
interface Hero {
        Good feat (Evil e);
}
```

```
class Superman implements Hero {
        public SuperGood hit (Evel e) {
                // save earth
                return new SuperGood(...);
        }
}
```

```
class abstract MarvelHero
        implements Hero {}
```

DonRiver

Интерфейсы

- Интерфейсы реализуются (имплементируются) с помощью ключевого слова implements
- Класс обязан реализовывать все методы либо являться абстрактным (до Java 8)

```
interface Hero {
        Good feat (Evil e);
}
```

```
class Superman implements Hero {
        public SuperGood hit (Evel e) {
                // save earth
                return new SuperGood(...);
        }
}
```

```
class abstract MarvelHero
        implements Hero {}
```

DonRiver

Интерфейсы. Константы

- Интерфейсы позволяют определять поля, которые будут константами

```
interface CommonAnswers{
        public static final String YES=”Yes”;
        public static final String NO=”No”;
        public static final String MAY_BE=”May be”;
}
```

Интерфейсы. Константы

- Интерфейсы позволяют определять поля, которые будут константами

```
interface CommonAnswers{
        String YES=”Yes”;
        String NO=”No”;
        String MAY_BE=”May be”;
}
```

Интерфейсы. Константы

- Интерфейсы позволяют определять поля, которые будут константами

```
interface CommonAnswers{
        String YES=”Yes”;
        String NO=”No”;
        String MAY_BE=”May be”;
}
```

Плохая практика!

DonRiver

Интерфейсы. Константы

- Интерфейсы позволяют определять поля, которые будут константами

```
public final class CommonAnswers{
        public static final String YES=”Yes”;
        public static final String NO=”No”;
        public static final String MAY_BE=”May be”;

        private CommonAnswers(){}
}
```

DonRiver

Интерфейсы. Расширение

- Интерфейсы могут расширять другие интерфейсы, используя ключевое слово extends

```
interface Animal {
        void eat();
}

interface Dog extends Animal {
        int bite(Animal a);
}
```

Интерфейсы. Методы по умолчанию

- Интерфейсы позволяют определять “методы по умолчанию”

```
public interface Person{
        default String live(int years) {
            return "I'v been living for " + years;
        }
}
```

DonRiver

Интерфейсы. Методы по умолчанию

- Интерфейсы позволяют определять “методы по умолчанию”

```
interface Animal{
        default String live(int years) {
            return "I'v been living for " + years;
        }
}
```

Интерфейсы. Методы по умолчанию

Q:
Зачем?

A:
Эволюция проекта влечет за собой изменения интерфейсов, что требует изменения ВСЕХ реализаций!

Методы по умолчанию решают эту проблему позволяя расширять интерфейсы, не меняя реализаций!

DonRiver

Интерфейсы. Методы по умолчанию

- Приоритет всегда отдается реализации метода в классе, а не методу по умолчанию
- Нельзя иметь одинаковые методы по умолчанию в 2 интерфейсах, от которых наследуется класс, не реализуя этот метод в нем

Интерфейсы. Методы по умолчанию

- Приоритет всегда отдается реализации метода в классе, а не методу по умолчанию
- Нельзя иметь одинаковые методы по умолчанию в 2 интерфейсах, от которых наследуется класс, не реализуя этот метод в нем
- Можно реализовать 2 интерфейса с одинаковыми методами по умолчанию, если реализовать их в классе!
- Можно сделать метод по умолчанию абстрактным, переобъявив его с ключевым словом abstract в расширенном интерфейсе или абстрактном классе!

Интерфейсы. Статические методы

- Интерфейс может определять один или несколько статических методов

```
interface TimeClient{
        static TimeZone getTimeZone(String zoneId) {
            TimeZone tz = /*convert zoneId here*/;
            return tz;
        }

        static TimeClient getTimeClient(String zoneId){
            return TimeClientFactory
                        .getTimeClient(getTimeZone(zoneId));
        }
}
```

DonRiver

Анонимные классы

- Специальные выражения, позволяющие реализовывать и создавать экземпляры интерфейсов, без создания нового класса!

```
Collections.sort(collection, new Comparator<Person>() {
   public int compare(Person o1, Persono2) {
       return o1.age > o2.age ? 1 : (o1.age < o2.age ? -1 : 0);
   }
});
```

Анонимные классы

Q:
Зачем?

Анонимные классы

Q:
Зачем?

A:
Удобное средство, для создания функций обратного вызова, отложенного вызова

Анонимные классы

Q:
Зачем?

A:
Удобное средство, для создания функций обратного вызова, отложенного вызова
если требуется доступ к локальному контексту (переменным, параметрам метода)

Анонимные классы

- Не имеют имени класса! Не могут иметь конструкторов!
- Имеют доступ к полям внешнего класса и к final переменным внешнего метода

Анонимные классы

- Не имеют имени класса! Не могут иметь конструкторов!
- Имеют доступ к полям внешнего класса и к final переменным внешнего метода (начиная с jdk 8 достаточно лишь завершенности переменной!)

Home Work

- Спроектировать интерфейс коллекции, с основными CRUD (create-read-update-delete) операциями
- Создать 2 реализации (на основе массива и на основе двусвязного списка)

- Написать утилитные методы для демонстрации работы с ними через интерфейс (использовать полиморфизм)