Программирование на языке Java

Основы ООП

Шумков Денис

Задача

Вывести клички всех кошек с возрастом больше одного года.

Решение 1

```
String firstCatName = "Барсик";
int firstCatAge = 3;
String secondCatName = "Барсик";
int secondCatAge = 5;
String thirdCatName = "Барсик";
int thirdCatAge = 1;

if (firstCatAge > 1) { System.out.println(firstCatName); }
if (secondCatAge > 1) { System.out.println(secondCatName); }
if (thirdCatAge > 1) { System.out.println(thirdCatName); }
```

Решение 2

```
String catName[] = new String[3];
int catAge[] = new int[3];
catName[0] = "Барсик"; catAge[0] = 3;
catName[1] = "Барсик"; catAge[1] = 5;
catName[2] = "Барсик"; catAge[2] = 1;

for (int i = 0; i <3; i++) {
    if (catAge[i] > 1) {
        System.out.println(catName[i]);
}
```

Что такое ООП?

- Концепция ООП предлагает оперировать в программе не переменными и функциями, а объектами.
- Всё в программе является объектами
- У объекта имеются свойства и методы
- Свойства представляют собой переменные, принадлежащие объекту
- Методы – функции, позволяющие получить / изменить информацию об объекте

Основные понятия ООП

- Абстракция
- Класс
- Объект
- Инкапсуляция
- Наследование
- Полиморфизм

Абстракция

Выделение значимых характеристик объекта, доступных остальной программе.

Абстракция кота

Абстракция кота

Значимые свойства

- Кличка
- Порода
- Цвет
- Рост
- Возраст
- Дата последнего кормления
- Дата последнего мяукания

Незначимые свойства

- Количество блох
- Громкость мурлыкания

Абстракция кота

Значимое поведение

- Мяукнуть
- Поесть
- Потребовать погладить
- Погулять

Незначимое поведение

- Рвать обои
- Испортить тапки

Абстракция кота

```
class Cat {
    String name;
    int age;
    Date lastMeowDate;

    void meow() {
        System.out.println("Meow!");
        lastMeowDate = new Date();
    }
}
```

Класс

Описываемая на языке терминологии исходного кода модель ещё не существующей сущности (объекта).

Инкапсуляция

Свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе.

Объект

Сущность в адресном пространстве вычислительной системы, появляющаяся при создании экземпляра класса (инстанцировании).

Кот – класс

Кот Барсик - объект

Объект

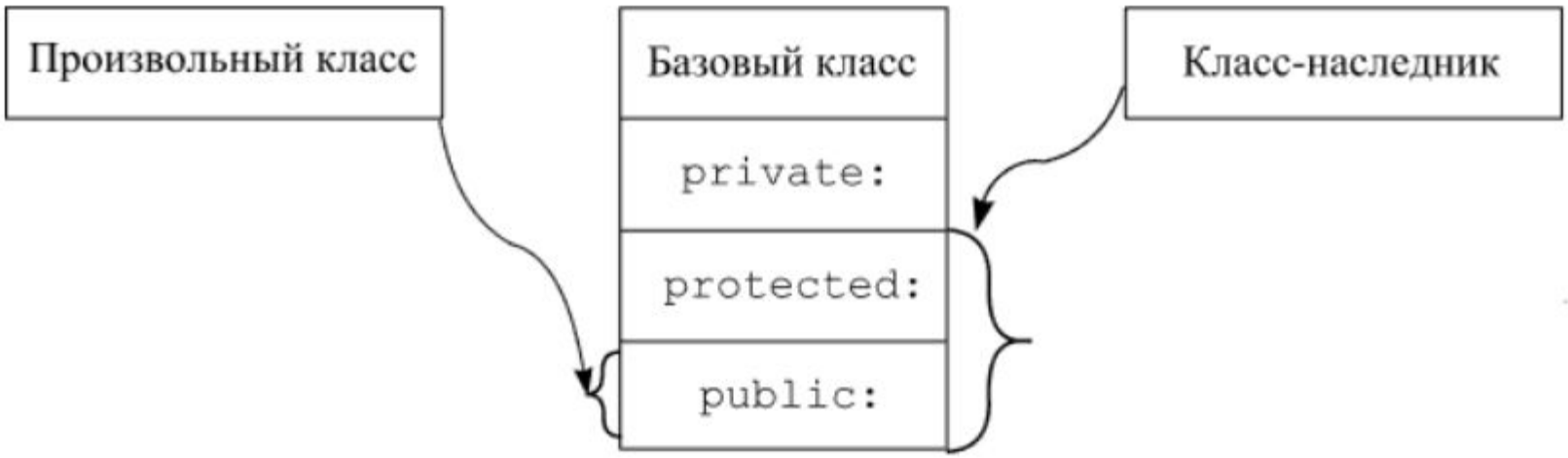
```
Cat cat = new Cat();
cat.name = "Барсик";
cat.meow();
```

Сокрытие свойств: зачем?

```
Cat cat = new Cat();
cat.name = "Барсик";
cat.age = -1;
```

Модификаторы доступа

- private: члены класса доступны только внутри класса;
- «default» (package-private) (модификатор, по-умолчанию): члены класса видны внутри пакета;
- protected: члены класса доступны внутри пакета и в наследниках;
- public: члены класс доступны всем;



Модификаторы доступа

- static - ссылка этого поля у любого экземпляра класса будет ссылаться на одно и то же значение
- final – это модификатор, позволяющий объявлять константные поля в классе.

Доступ к сокрытым свойствам

Класс:

```
class Cat {
    private int age;
    public void setAge(int age) {
        if (age < 0) {
            shutdownPC();
        } else {
            this.age = age;
        }
    }
    public int getAge() {
        return age;
    }
}
```

Использование

```
Cat cat = new Cat();
cat.setAge(3);
System.out.println(cat.getAge());
```

Указатель this

Указатель на объект, из которого он был вызван.

Cat cat = **new** Cat();

cat.setAge(3); - В методе setAge указатель this будет ссылаться на объект cat

Задание 1

Решить задачу с выборкой кошек по возрасту, используя ООП.

Перегрузка (Overloading)

Создание метода с таким же именем, но с другим набором параметров.

Перегрузка методов

```
public class Cat {
    public void eat() {}
    public void eat(int size) {
        for (int i = 0; i < size; i++) eat();
    }
    public void eat(Food food) {}
    public void eat(Food food, int size) {
        for (int i = 0; i < size; i++) eat(food);
    }
}
```

Конструкторы

```
public class Cat {
    private String name;
    private int age;

    public Cat() {
        name= "Unnamed";
        age = 0;
    }
    public Cat(String name) {
        this.name = name;
        age = 0;
    }
    public Cat(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

```
public class Cat {
    private String name;
    private int age;

    public Cat() {
        this(“Unnamed”);
    }
    public Cat(String name) {
        this.name = name;
        age = 0;
    }
    public Cat(String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

Конструктор vs Метод

Конструктор

```
class Cat {

    Cat() {
    }

}
```

Метод

```
class Cat {

    void Cat() {
    }

}
```

Задание 2

Модифицировать задачу с выборкой кошек по возрасту, используя конструкторы.

Абстракция собаки

Собака vs Кот: Свойства

Кот

- Кличка
- Порода
- Цвет
- Рост
- Возраст
- Дата последнего кормления
- **Дата последнего мяукания**

Собака

- Кличка
- Порода
- Цвет
- Рост
- Возраст
- Дата последнего кормления
- **Дата последнего гавкания**

Собака vs Кот: Методы

Кот

- **Мяукнуть**
- Поесть
- Потребовать погладить
- **Погулять**

Собака

- **Гавкнуть**
- Поесть
- Потребовать погладить
- **Выгуляться**

Домашние животные

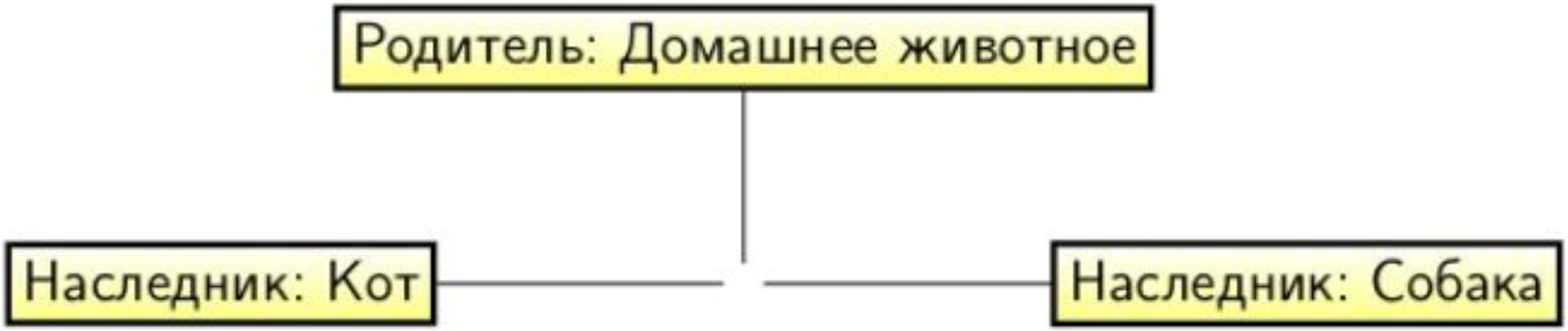
Наследование

Свойство системы, позволяющее описать новый класс на основе уже существующего с частично или полностью заимствующейся функциональностью. Класс, от которого производится наследование, называется **базовым**, **родительским** или **суперклассом**. Новый класс — **потомком**, **наследником**, **дочерним** или **производным классом**.

Принцип наследования

Общие свойства и методы объектов можно вынести в класс-**«родитель»**. Все **«дети»**-наследники автоматически получают их.

Схема наследования



Класс домашних животных

```
public class Pet {
    private String name;
    private int age;
    public Pet() { this("Unnamed");}
    public Pet(String name) {name = "Unnamed";}
    public Pet(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void setName(String name) { this.name = name; }
    public String getName() { return name; }
    public void requireToPat() {}
    public String getType() { return "Домашнее животное"; }
}
```

Класс кота

```
public class Cat extends Pet {
    private Date lastMeowDate;
    public Cat() { this("Unnamed"); }
    public Cat(String name) { super(name); }
    public Cat(String name, int age) { super(name, age); }
    @Override
    public String getType() { return "Кот"; }
    public void meow() {
        System.out.println("Meow!");
        lastMeowDate = new Date();
    }
}
```

super()

Ссылка на базовый класс, которую можно использовать в дочерних классах

Переопределение (Overriding)

Переписывание (переделывание, переопределение) в классе-потомке УЖЕ существующего метода класса-родителя.

Аннотация Java

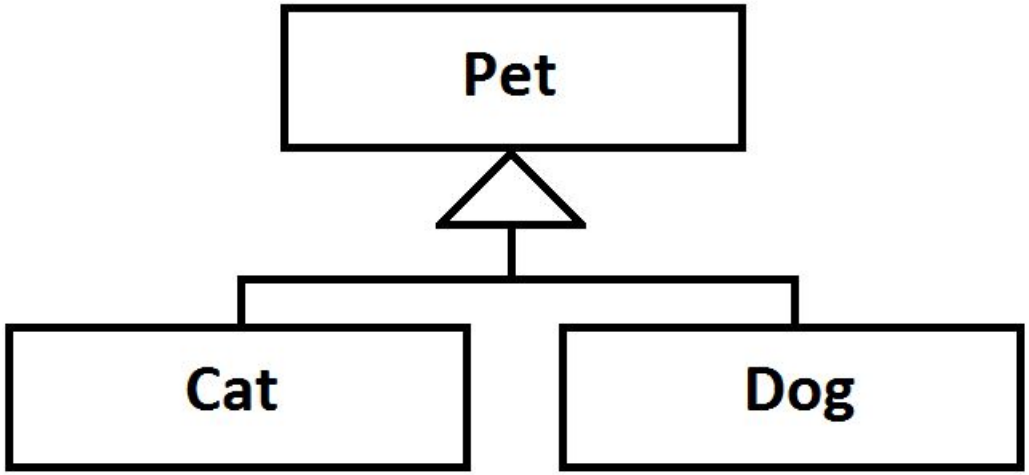
Специальная форма синтаксических метаданных, которая может быть добавлена в исходный код. **Аннотации** используются для анализа кода, компиляции или выполнения. Аннотируемы пакеты, классы, методы, переменные и параметры.

Аннотация @Override

Проверяет, переопределён ли метод.
Вызывает ошибку компиляции, если метод не найден в родительском классе;

UML – диаграмма классов

UML = Unified Modeling Language



Полиморфизм

Свойство системы, позволяющее использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Полиморфизм

```
Pet pet1 = new Pet();
Pet pet2 = new Cat();
Pet pet3 = new Dog();
pet1.getType();
pet2.getType();
pet3.getType();
```

Тип ссылки vs Тип объекта

```
Pet pet1 = new Pet();
Pet pet2 = new Cat();
Pet pet3 = new Dog();
pet1.getType();
pet2.getType();
pet2.meow();
pet3.getType();
pet3.woof();
```

Приведение типов

```
Pet pet1 = new Pet();
Pet pet2 = new Cat(); //автоматическое
Pet pet3 = new Dog(); //автоматическое
Cat cat = (Cat) pet2; cat.meow(); //явное
((Cat) pet2).meow(); //явное
Dog dog = (Dog) pet3; dog.woof(); //явное
((Dog) pet3).woof();  //явное
```

Есть ли ошибки?

```
Pet pet = new Pet();
Cat cat = new Cat();
Dog dog = new Dog();
```

```
(Pet) pet;
(Pet) cat;
(Pet) dog;
```

```
(Cat) pet;
(Cat) cat;
(Cat) dog;
```

```
(Dog) pet;
(Dog) cat;
(Dog) dog;
```

Конечно есть!

```
Pet pet = new Pet();
Cat cat = new Cat();
Dog dog = new Dog();
```

(Pet) pet;	**(Cat) pet;**	**(Dog) pet;**
(Pet) cat;	(Cat) cat;	**(Dog) cat;**
(Pet) dog;	**(Cat) dog;**	(Dog) dog;

Типы ошибок в Java

Ошибка компиляции

(Cat) dog;

(Dog) cat;

Ошибка времени выполнения

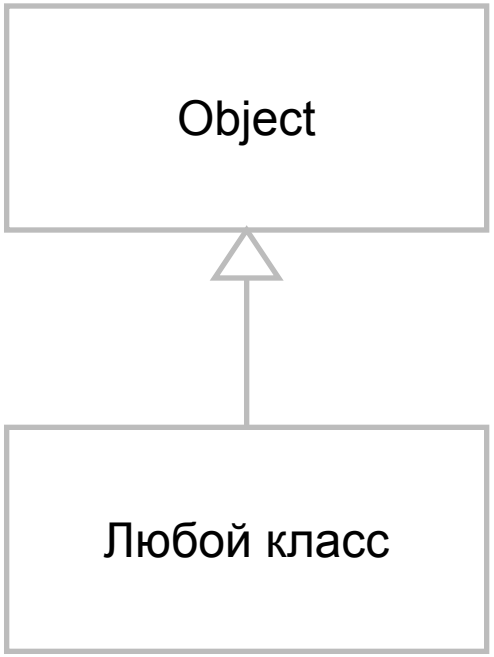
(Cat) pet;

(Dog) pet;

Задание 3

Модифицировать задачу с выборкой кошек по возрасту, добавив в изначальный список собак. К выводу клички добавить тип животного («кот Барсик», «собака Улыбака»).

Класс Object



Методы Object

- **boolean** equals(Object obj);
- String toString();
- **int** hashCode();
- Object clone();

Задание 4

Создать класс комплексных чисел

```
public class Complex {
    private int re;
    private int im;
    Complex(int re, int im) { … }
    @Override
    public String equals(Object obj) { … }
    @Override
    public String toString() { … }
    // Метод, реализующий операцию сложения
    public Complex plus(Complex z) { … }
    // Метод, реализующий операцию вычитания
    public Complex minus(Complex z) { … }
}
```

Задача 4

Создать класс для демонстрации работы класса комплексных чисел:

```
public class ComplexTest {
    public static void main(String[] args) {
        Complex z1 = new Complex(1, 0);
        Complex z2 = new Complex(2, 3);
        Complex z3 = new Complex(5, 10);
        System.out.println("z1 = " + z1);
        System.out.println("z2 = " + z2);
        System.out.println("z3 = " + z3);
        System.out.println("z1 + z2 = " + z1.plus(z2));
        System.out.println("z1 - z3 = " + z1.minus(z3));
        System.out.println(z1.equals(z3) ? “z1 == z3” : “z1 != z3”);
    }
}
```

Абстрактный класс

Базовый класс, который не предполагает создания экземпляров.

Абстрактное животное

```
public abstract class Pet {
    private String name;
    private int age;
    public Pet() { this("Unnamed");}
    public Pet(String name) {name = "Unnamed";}
    public Pet(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void setName(String name) { this.name = name; }
    public String getName() { return name; }
    public String getType() { return "Домашнее животное"; }
    public abstract void say();
}
```

Конкретный кот

```
public class Cat extends Pet {
    private Date lastMeowDate;
    public Cat() { this("Unnamed"); }
    public Cat(String name) { super(name); }
    public Cat(String name, int age) { super(name, age); }
    @Override
    public String getType() { return "Кот"; }
    @Override
    public void say() {
        System.out.println("Meow!");
        lastMeowDate = new Date();
    }
}
```

Интерфейс

Конструкция в коде программы, используемая для описания совокупности возможностей, предоставляемых классом или компонентом.

Интерфейс автомобиля

```
interface Car {
    KPP getKPP();
    Airbag getAirbag();
    void beep();
    void runEngine(Engine engine);
}
```

Реализация интерфейса

Бумер

```
class BMW implements Car {
    public KPP getKPP() {
        return new AutoKPP;
    }
    public Airbag getAirbag() {
        return new VerySafeAirbag();
    }
}
```

Лада Седан Баклажан

```
class LADA implements Car {
    public KPP getKPP() {
        return new ManualKPP;
    }
    public Airbag getAirbag() {
        return null;
    }
}
```

Правило наследования Java

Класс-наследник может наследовать только один базовый класс, но множество интерфейсов.

Абстракция женщины-кошки

Что должна уметь женщина-кошка?

Интерфейс женщины

```
interface Woman {
   void seduce ();
   void beBeautiful();
}
```

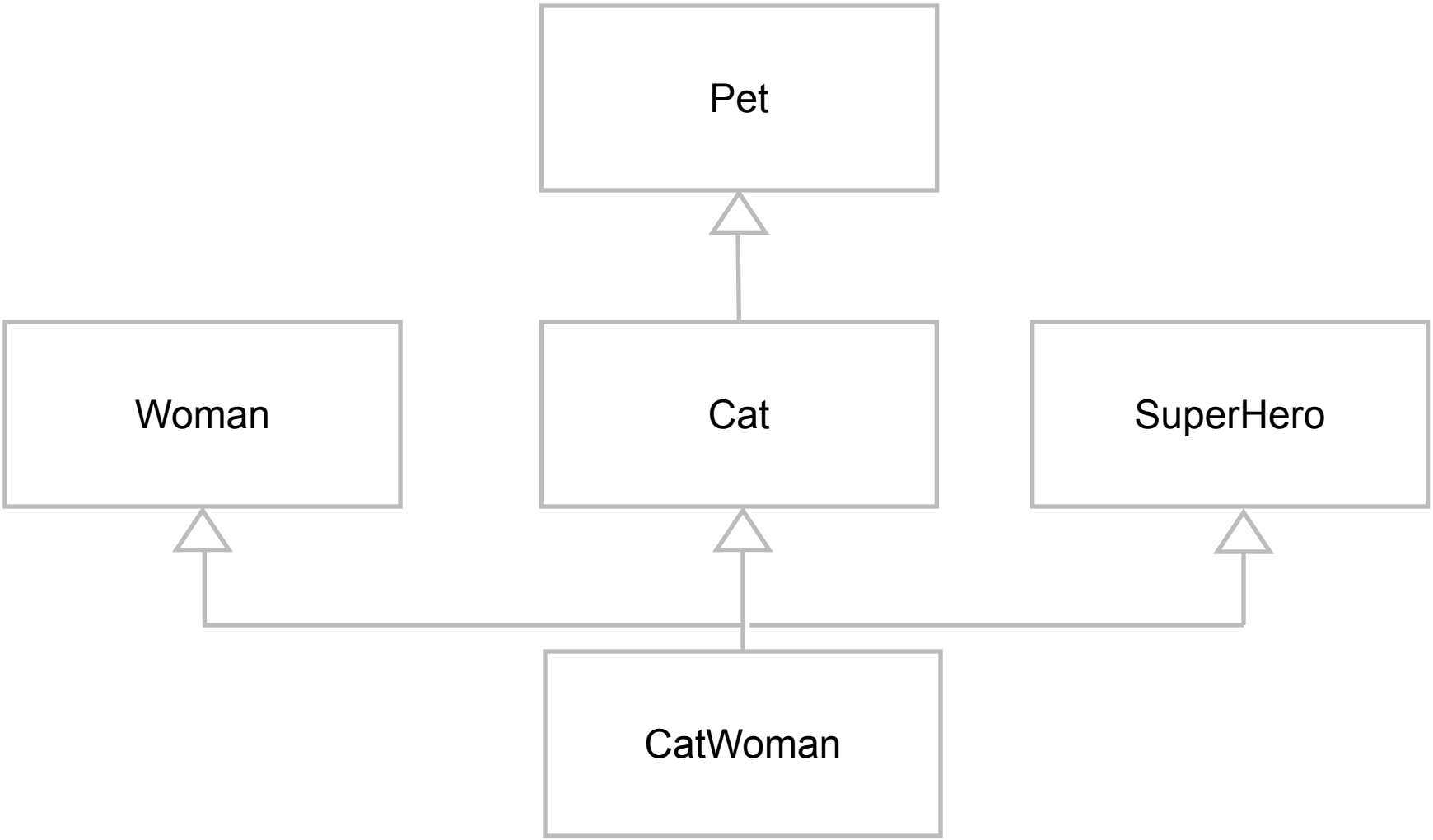
Интерфейс супергероя

```
interface SuperHero {
    Costume getCostume();
    SuperPower getSuperPower();
}
```

Реализуем женщину-кошку

```
class  CatWoman extends Cat implements
Woman, SuperHero {
    void seduce () {…}
    void beBeautiful() {…}
    Costume getCostume() {…}
    SuperPower getSuperPower() {…}
}
```

UML-диаграмма женщины-кошки



Задача 5

Реализовать интерфейс стека, работающего с символами:

```
interface IStackChar {
    void push(char ch);
    char pop();
    boolean isEmpty();
}
```

Реализованный класс назвать StackChar. Создать класс StackCharTest для теста класса StackChar.

Задача 6

Создать класс DequeChar, унаследованный от класса StackChar. Добавить в него операции pushToHead и popFromHead. Создать класс DequeCharTest для теста класса DequeChar.

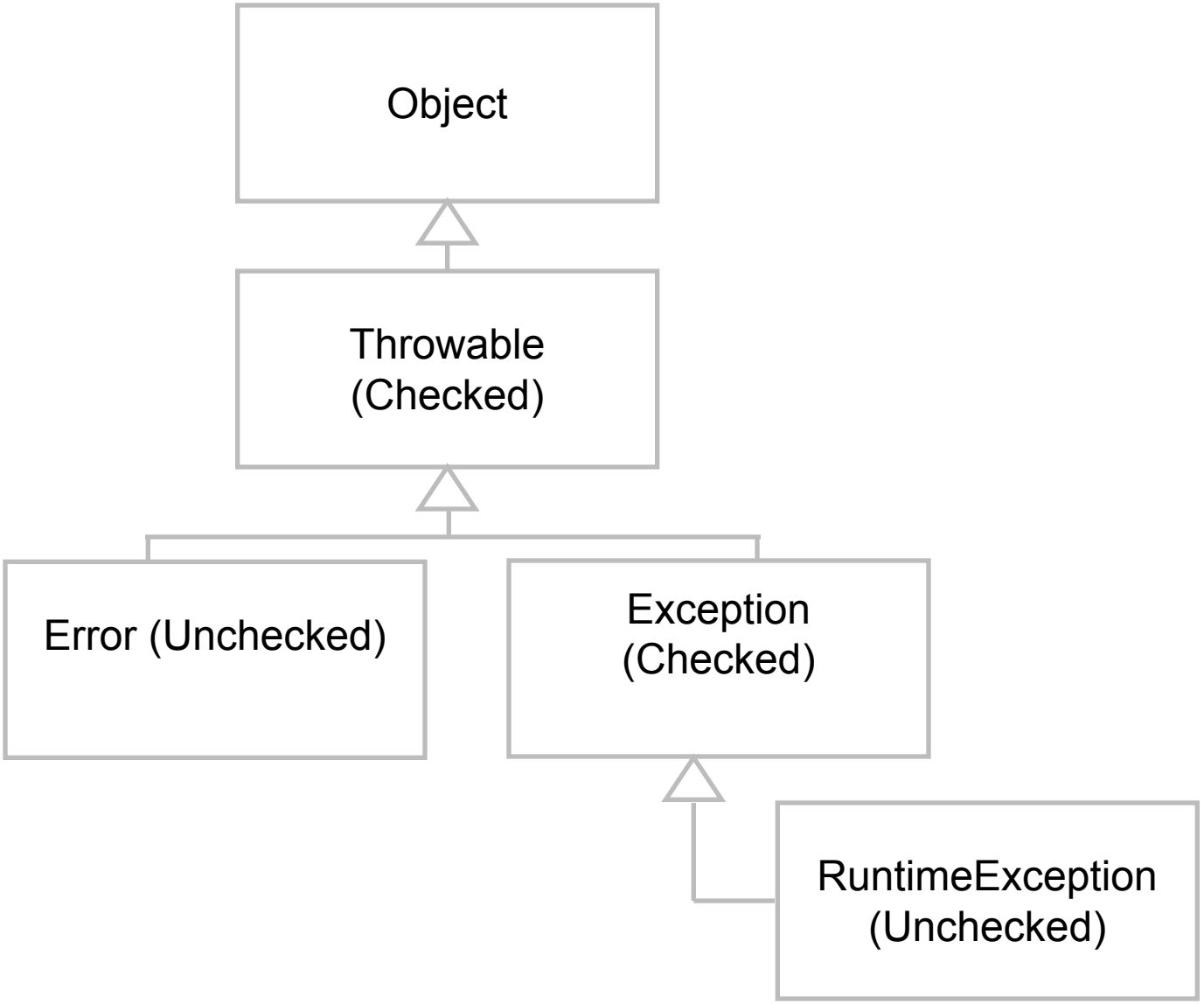
Исключения Java

```
try {
        Код, который может бросать исключения
} catch (Exception e) {
    Обработка перехваченного исключения
} final {
    Блок выполняется либо после try,
    либо после  catch
}
```

Исключения Java

```
int a = 5;
int b = 0;
try {
        float c = a / b;
} catch (ArithmeticException e) {
    System.out.println("Делить на ноль нельзя!");
}
```

Исключения Java



Домашнее задание

Создать приложение с классом и тестом этого класса. Использовать Java Code Convention.

В каждом варианте требуется сделать:

1. Класс X
2. Класс Xtest

Класс Xtest содержит ТОЛЬКО метод main() в котором создается 2-3 объекта типа X. И по очереди вызываются ВСЕ методы класса X с выводом результатов в консоль.

Класс X содержит конструктор с параметрами, и конструктор по умолчанию (без параметров). Также X содержит методы, подразумеваемые Вашим вариантом.

Домашнее задание

Варианты:

1. *Класс Котопёс (CatDog)*
2. *Класс Стол (Table)*
3. *Класс Книга (Book)*
4. *Класс Автомобиль (Car)*
5. *Класс Компьютер (Computer)*
6. *Класс Путешествие (Trip)*
7. *Класс Город (City)*
8. *Класс Пассажир (*Commuter*)*
9. *Класс Студент (Student)*
10. *Класс рациональное число (Rational)*