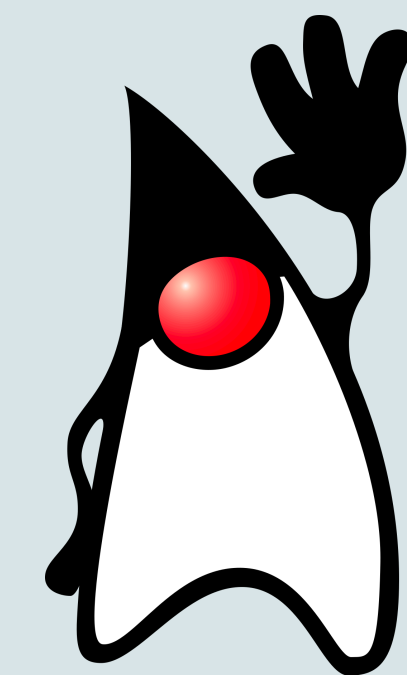




# СБЕРБАНК

---

Корпоративный  
университет



# Основы разработки

Занятие № 2

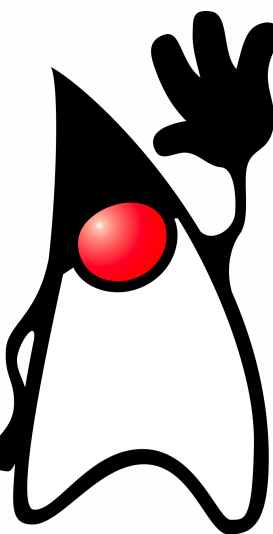


**СБЕРБАНК**

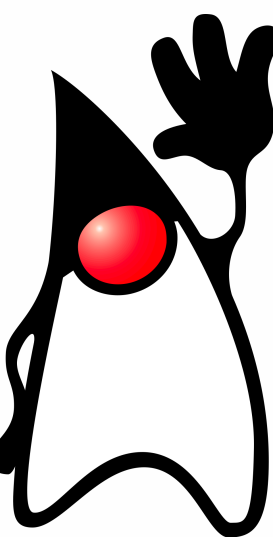
Корпоративный  
университет



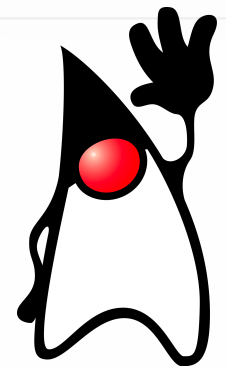
- Quiz. Командная строка, ключевые команды
- Основные концепции программирования
- ООП в Java. Пакеты, классы, методы, параметры
- Типы данных в Java
- Generic
- Collection ???



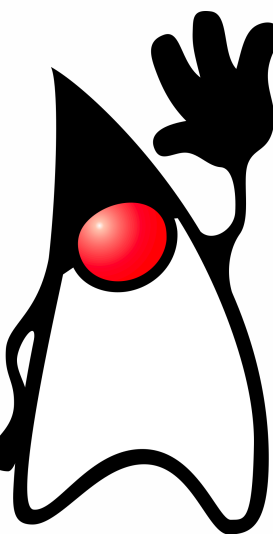
- Активно участвуем. Не стесняйтесь задавать вопрос.
- Но off-topic обсуждаем в Telegram @sb\_ku\_java\_2019\_10
- Не стесняйтесь просто спрашивать в telegram.
- В конце с Вас отзыв.
- ДЗ - Будет !



**Договорились?  
Поехали!**



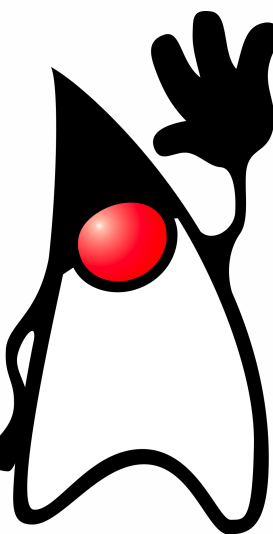
- **Quiz. Командная строка, ключевые команды**
- Основные концепции программирования
- ООП в Java. Пакеты, классы, методы, параметры
- Типы данных в Java
- Первая поставка Java-артефакта. Хранение собранных артефактов
- Maven, продвинутые фишки



00

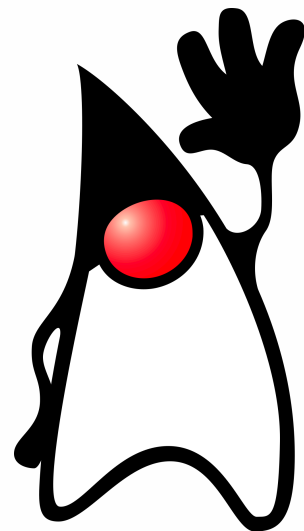
# Quiz. Командная строка

- Bash - \*nix (Unix, Linux, \*BSD, MacOS)
- Shell - Windows





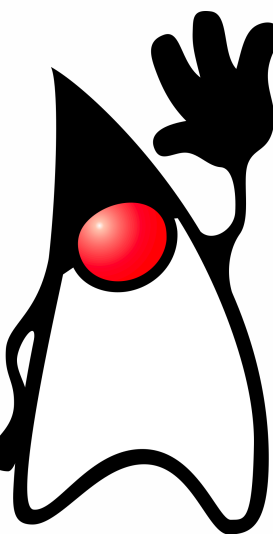
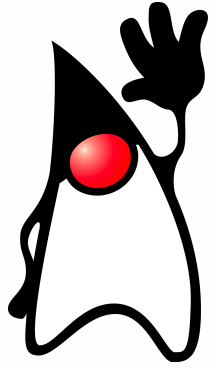
	Bash	Shell
Просмотр содержимого каталога	ls	dir
Переход в каталог	cd	cd
Разделитель в полном пути к файлу	/	\
Регистрозависимость	+	-
Скрипты для команд командной строки	.sh	.bat
Подключение к удаленному терминалу	+	- (но есть Putty)
Копирование файла	cp	xcopy
Обмен данными с удаленной машиной	scp	-



# Ваши вопросы?

Если что — их можно задать  
ПОТОМ

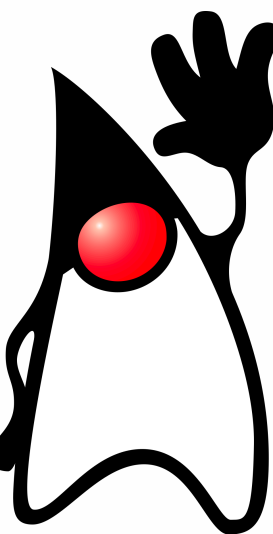
- Quiz. Командная строка, ключевые команды
- **Основные концепции программирования**
- ООП в Java. Пакеты, классы, методы, параметры
- Типы данных в Java
- Первая поставка Java-артефакта. Хранение собранных артефактов
- Maven, продвинутые фишки



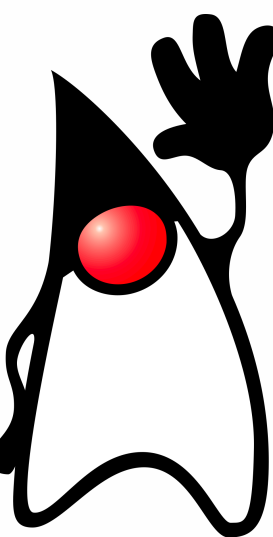
02

# Основные концепции программирования

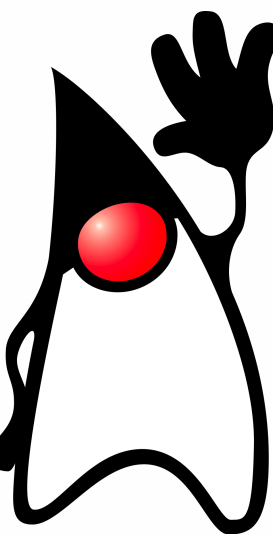
# Что такое программирование?



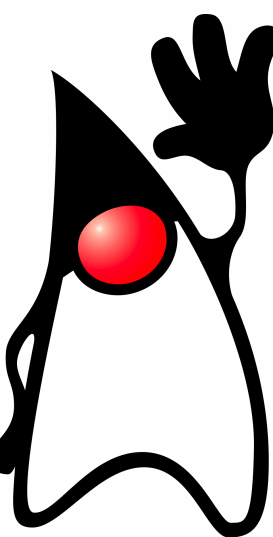
- Алгоритмическое программирование
- Процедурное программирование (Pascal, C, Basic, Fortran, Go...)
- Объектно-ориентированное программирование (C++, Java, Delphi, Visual Basic...)
- Аспектно-ориентированное программирование (AspectJ)
- Функциональное программирование (Lisp, Scala, Erlang, Haskell, F#...)
- Декларативное программирование (SQL, POM)



- **Абстракция**
- **Полиморфизм**
- **Инкапсуляция**
- **Наследование**



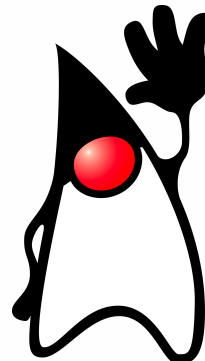
- **Абстракция** для выделения в моделируемом предмете важного для решения конкретной задачи по предмету, в конечном счёте — контекстное понимание предмета, формализуемое в виде класса
- **Полиморфизм** для определения точки, в которой единое управление лучше распараллелить или наоборот — собрать воедино.
- **Инкапсуляция** - свойство системы, позволяющее объединить данные и методы, работающие с ними, в классе. В Java совмещено с сокрытием
- **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.

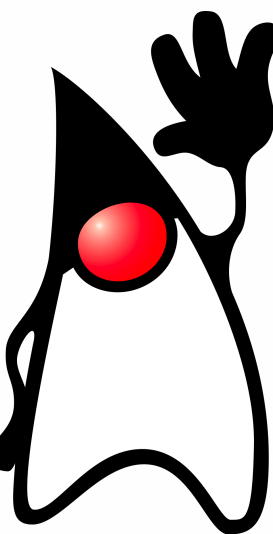




# Ваши вопросы?

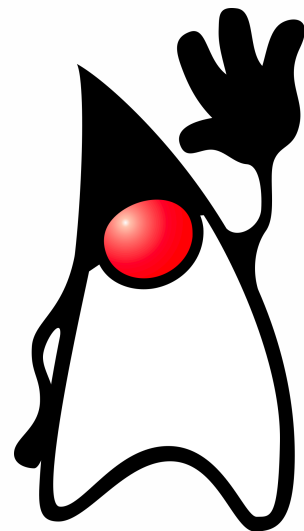
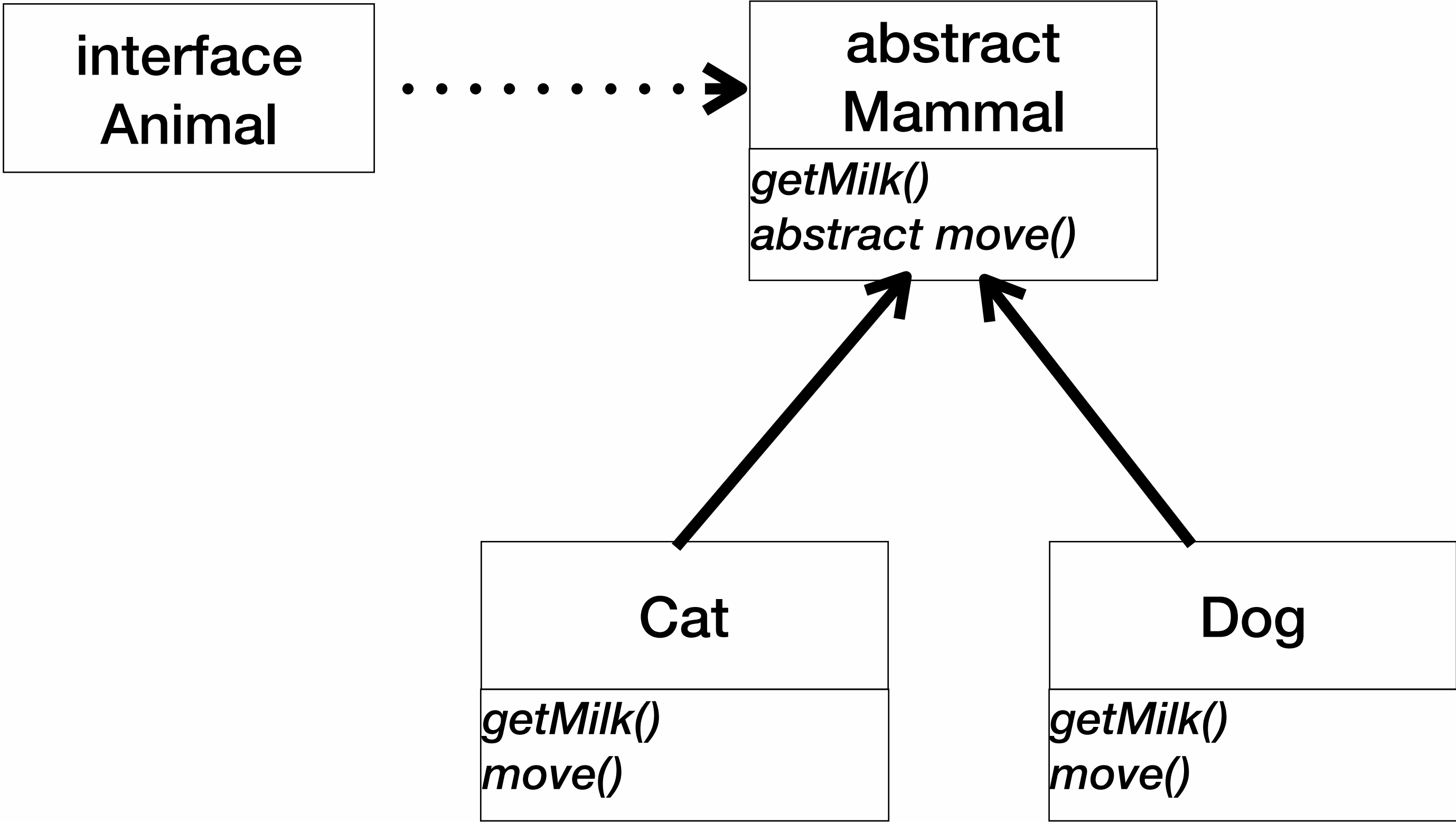
Если что — их можно задать  
ПОТОМ

- Quiz. Командная строка, ключевые команды
- Основные концепции программирования
-  • **ООП в Java. Пакеты, классы, методы, параметры**
- Типы данных в Java
- Generic
- Collection ???



03

**ООП в Java.**



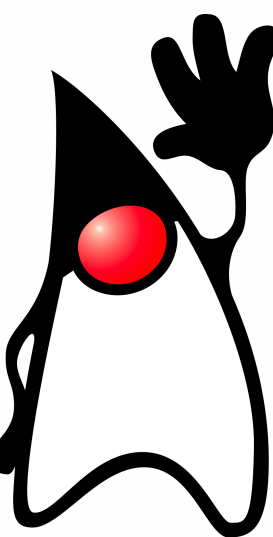
# Создание экземпляра класса

```
Cat cat = new Cat();
```

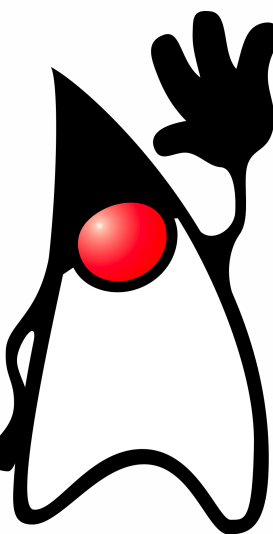
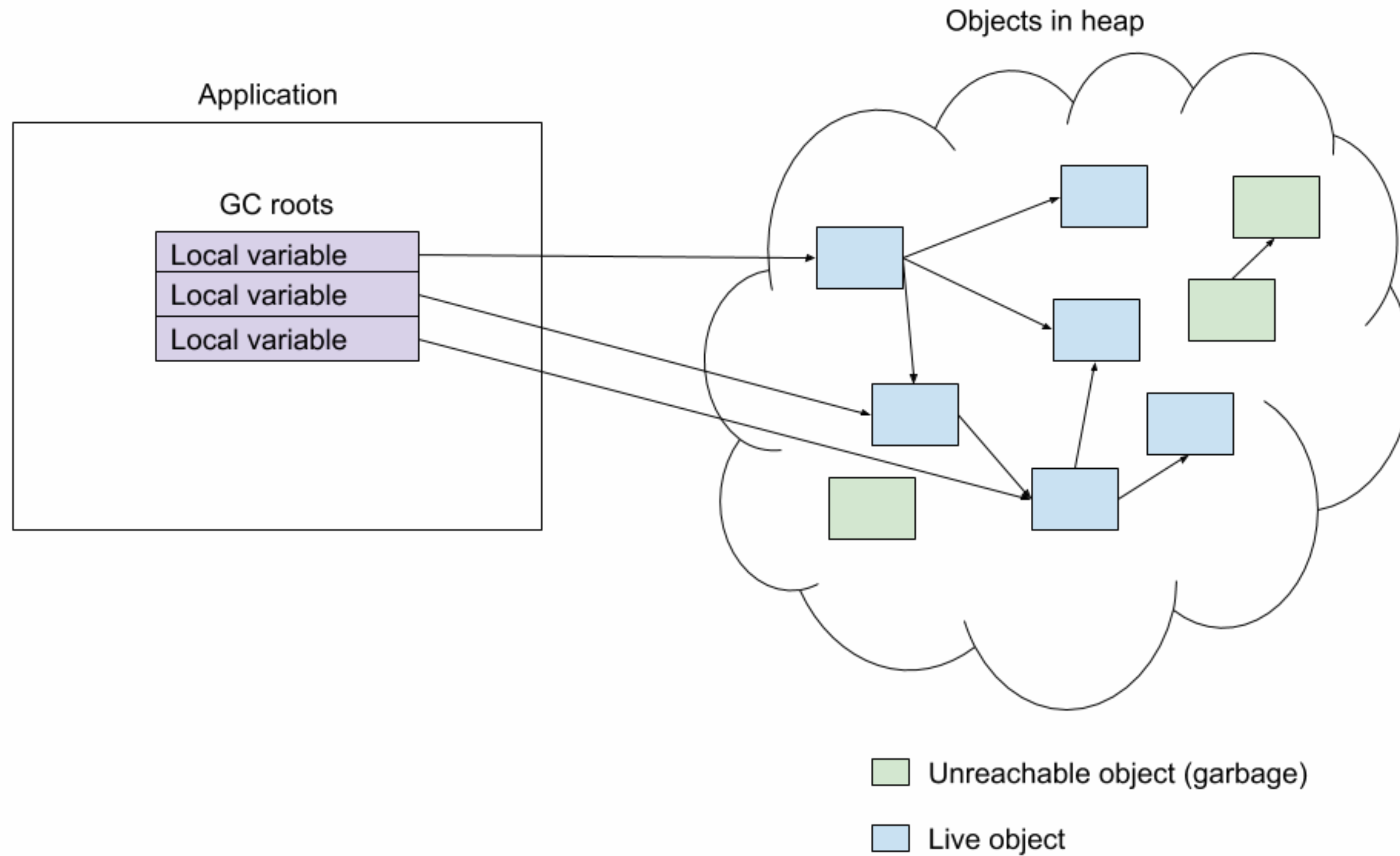
```
Animal animal = new Dog();
```

```
Animal animal = new Animal();
```

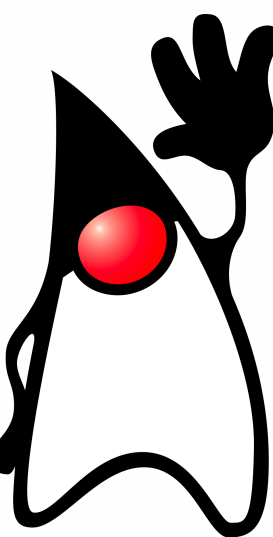
```
Mammal mammal = new Mammal();
```



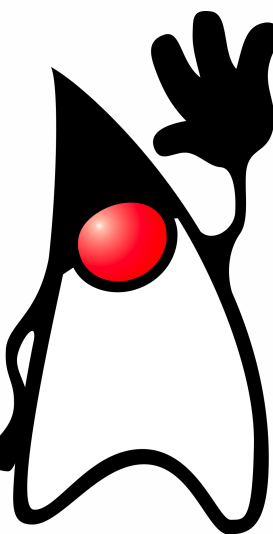
# Жизненный цикл экземпляра объекта. Garbage Collector



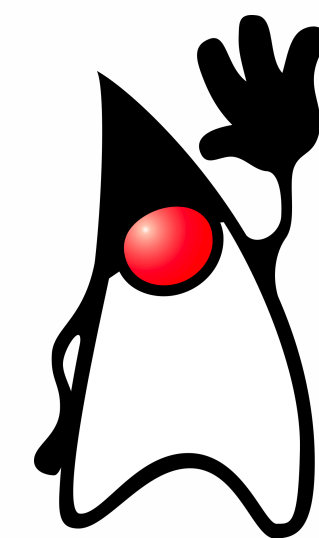
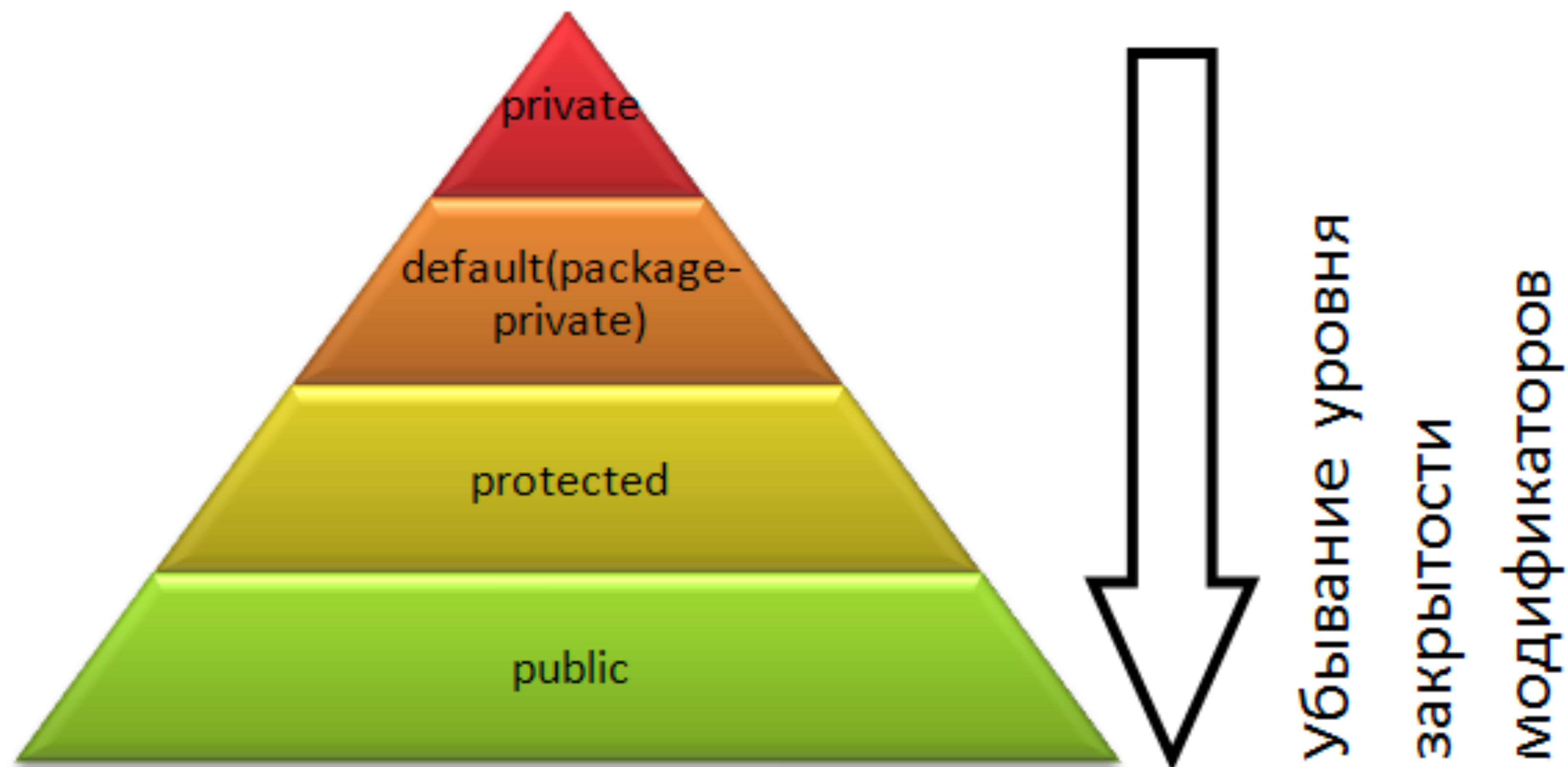
- `public class` должен находиться в файле с таким же названием
- !!!Java case sensitive!!!
- Имена классов с большой буквы в CamelCase
- Имена полей, методов - с маленькой буквы в CamelCase
- Константы - большими буквами, с подчеркиванием в разрыв слов
- Нельзя начинать имена с цифры (классы, методы, поля, константы)



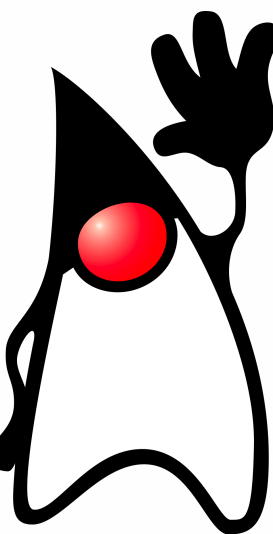
```
package ru.sbrf.ku.java.generics.bounds.entries;  
  
public class Animal {  
    @Override  
    public String toString() {  
        return "Animal";  
    }  
}
```



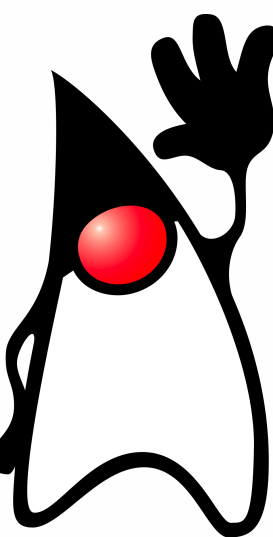




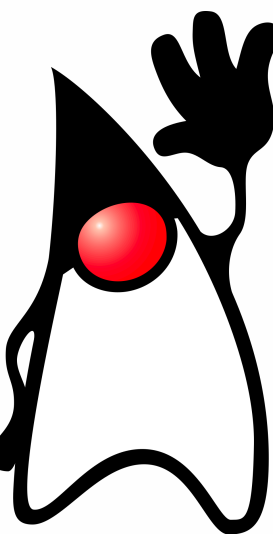
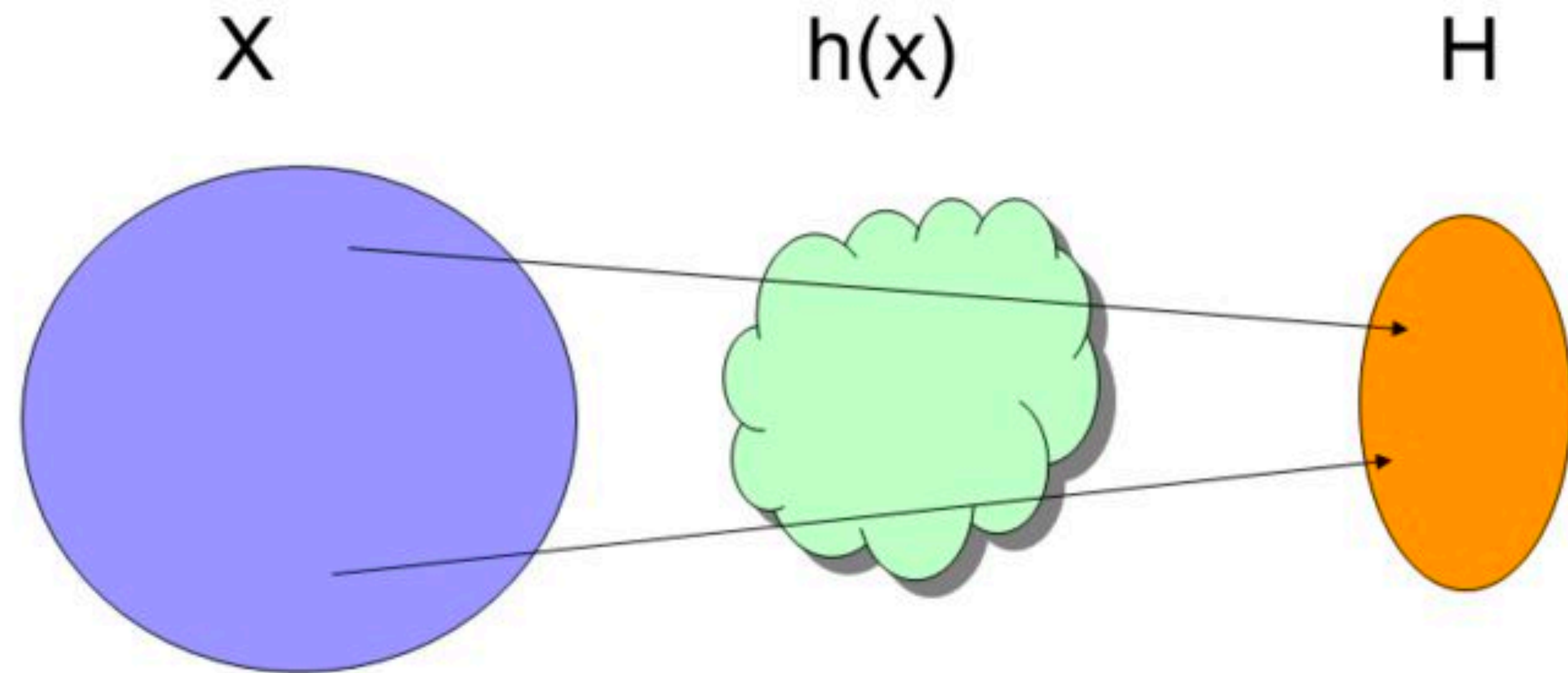
- **Класс (class)**
- **Интерфейс (interface)**
- **Перечисление (enum)**
- **\*Аннотации (@)**



```
public class Object {  
    public Object() {}  
  
    public final native Class<?> getClass();  
  
    public native int hashCode();  
  
    public boolean equals(Object obj) {  
        return (this == obj);  
    }  
  
    protected native Object clone() throws CloneNotSupportedException;  
  
    public String toString() {  
        return getClass().getName() + "@" + Integer.toHexString(hashCode());  
    }  
  
    public final native void notify();  
  
    public final native void notifyAll();  
  
    public final void wait() throws InterruptedException {  
        wait(0L);  
    }  
  
    ....  
}
```



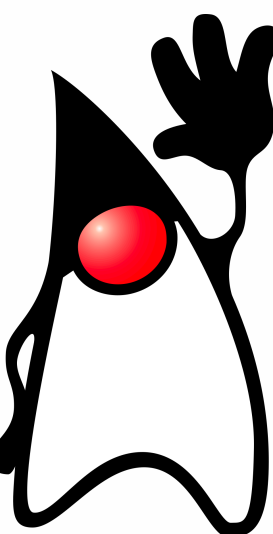
## Функция хеширования



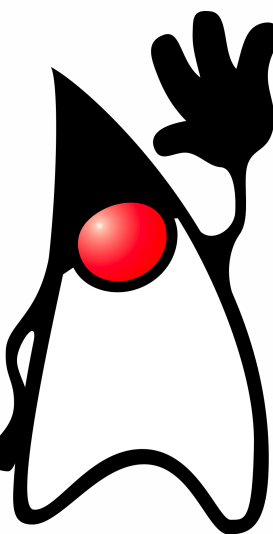
**Рефлексия** (от позднелат. reflexio — обращение назад) — это механизм исследования данных о программе во время её выполнения. Рефлексия позволяет исследовать информацию о полях, методах и конструкторах классов.

Вот основной список того, что позволяет рефлексия:

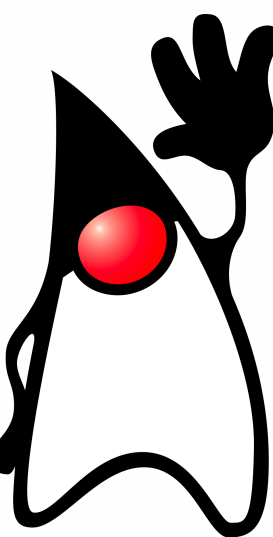
- Узнать/определить класс объекта;
- Получить информацию о модификаторах класса, полях, методах, константах, конструкторах и суперклассах;
- Выяснить, какие методы принадлежат реализуемому интерфейсу/интерфейсам;
- Создать экземпляр класса, причем имя класса неизвестно до момента выполнения программы;
- Получить и установить значение поля объекта по имени;
- Вызвать метод объекта по имени.



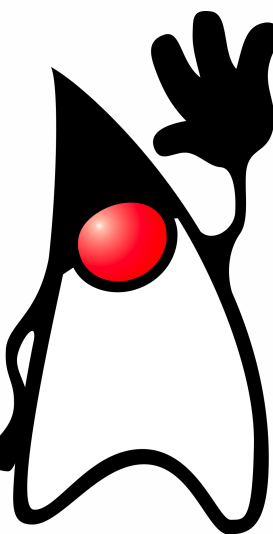
```
public class Intro {  
  
    public final static Integer SUCCESS_CODE = 200;  
  
    private static Integer code;  
  
    public static Integer getCode(){  
        return code;  
    }  
}
```



```
public class Dog {  
    private String name;  
  
    public String getName(){  
        return name;  
    }  
}  
  
public class Zoo {  
    public void test(){  
        Dog dog = new Dog();  
        System.out.println( dog.getName() );  
    }  
}
```



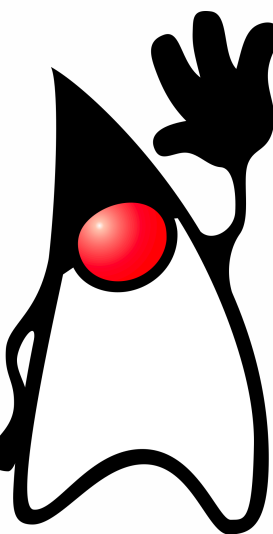
```
public class Dog {  
    private String name;  
  
    public Dog(){  
    }  
  
    public Dog(String name){  
        this.name = name;  
    }  
  
    public String getName(){  
        return name;  
    }  
}
```



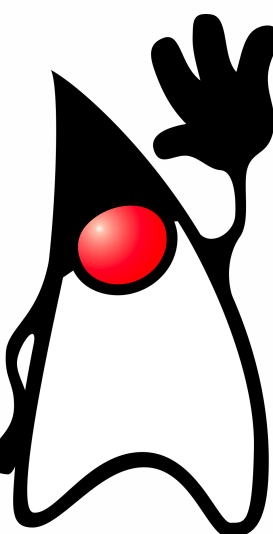


Внутренним классом называют класс, который является членом другого класса. Существует четыре базовых типа внутренних классов в Java:

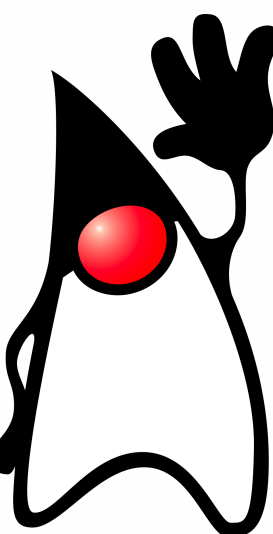
1. Nested Inner classes (вложенные внутренние классы)
2. Static Nested classes or Member of outer class (статические вложенные классы)
3. Method Local Inner classes (внутренние классы в локальном методе)
4. Anonymous Inner classes (анонимные классы)



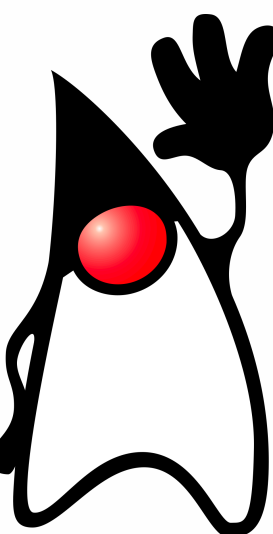
```
public class Outer {  
    // Простой вложенный класс  
    class Inner {  
        public void show() {  
            System.out.println("Метод внутреннего класса");  
        }  
    }  
  
    public static void main(String[] args) {  
        Outer.Inner inner = new Outer().new Inner();  
        inner.show();  
    }  
}
```



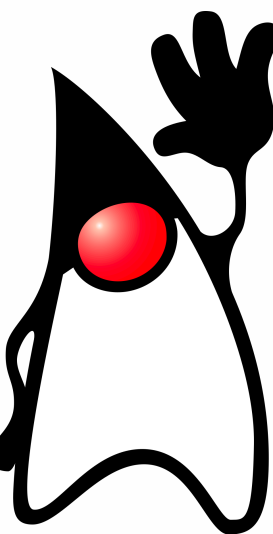
```
public class Outer {  
    // Статический внутренний класс  
    static class Inner {  
        public void show() {  
            System.out.println("Метод внутреннего класса");  
        }  
    }  
  
    public static void main(String[] args) {  
        Outer.Inner inner = new Outer.Inner();  
        inner.show();  
    }  
}
```

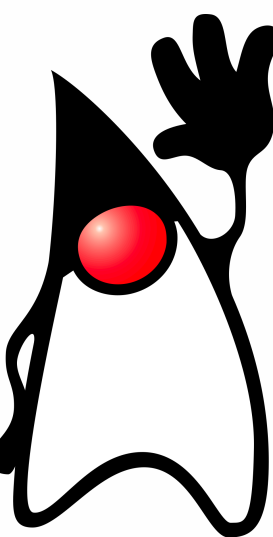
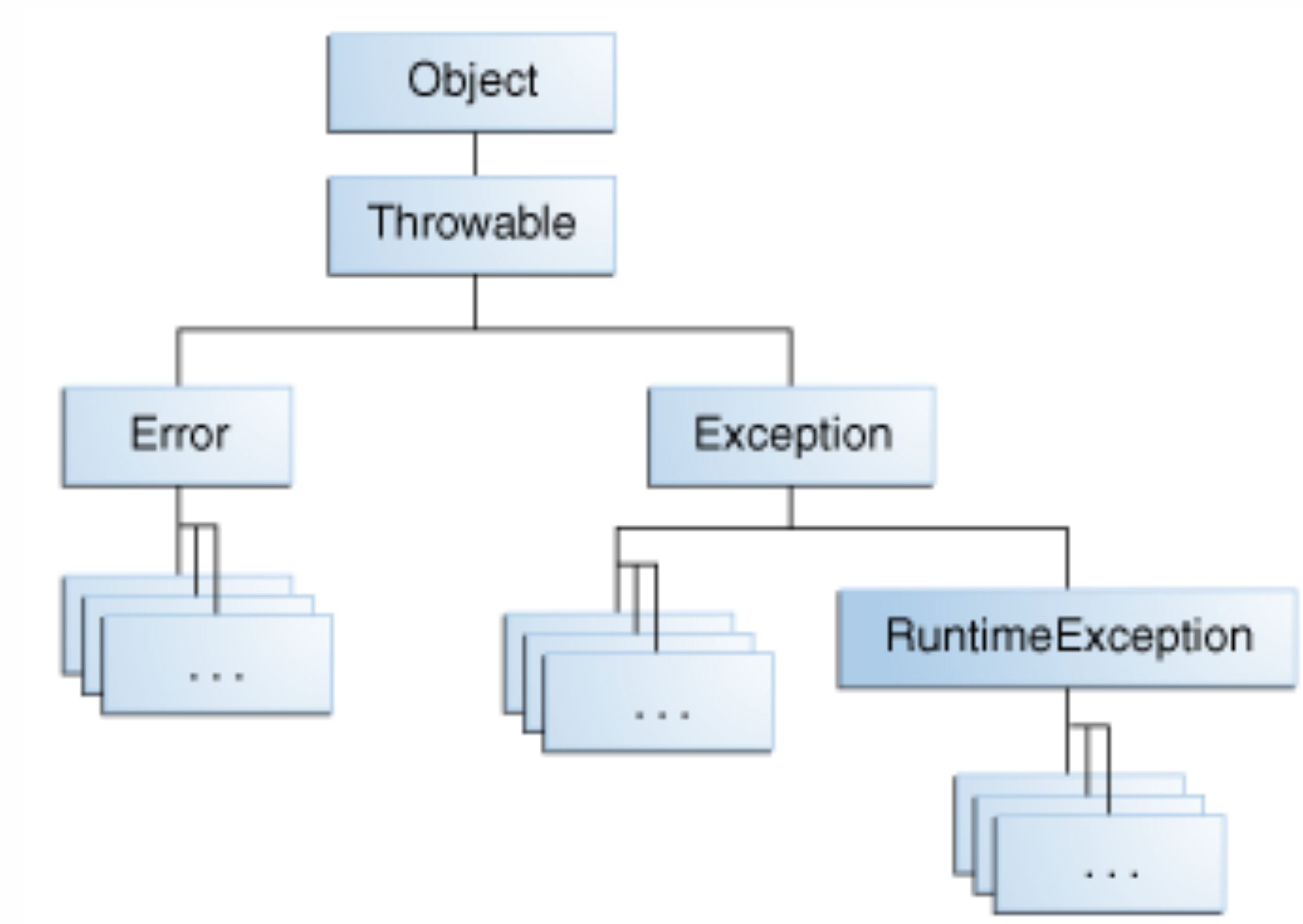


```
public class Outer {  
    void outerMethod() {  
        System.out.println("Метод внешнего класса");  
        // Внутренний класс является локальным для метода outerMethod()  
        class Inner {  
            public void innerMethod() {  
                System.out.println("Метод внутреннего класса");  
            }  
        }  
        Inner inner = new Inner();  
        inner.innerMethod();  
    }  
  
    public static void main(String[] args) {  
        Outer outer = new Outer();  
        outer.outerMethod();  
    }  
}
```



```
public class Outer {  
    // Анонимный класс, который реализует интерфейс Hello  
    static Hello h = new Hello() {  
        public void show() {  
            System.out.println("Метод внутреннего анонимного класса");  
        }  
    };  
  
    public static void main(String[] args) {  
        h.show();  
    }  
}  
  
interface Hello {  
    void show();  
}
```

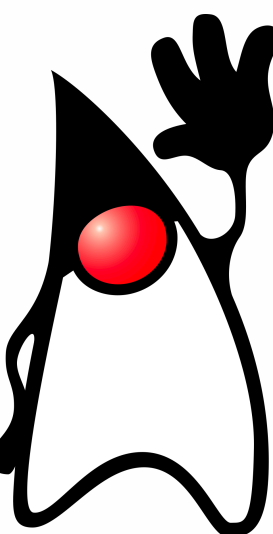
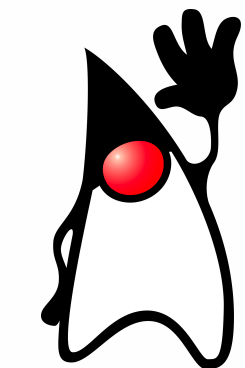




# Ваши вопросы?

Если что — их можно задать  
ПОТОМ

- Quiz. Командная строка, ключевые команды
- Основные концепции программирования
- ООП в Java. Пакеты, классы, методы, параметры
- **Типы данных в Java**
- Generic
- Collection ???



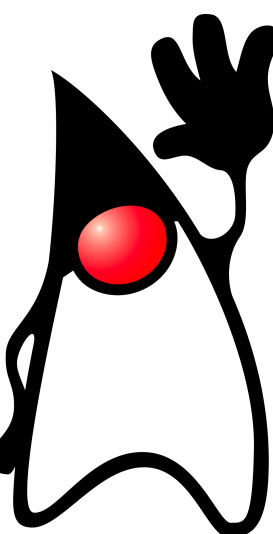


04

# Типы данных в Java

Скалярные (передаются по значению):

- byte (целые числа, 1 байт)
- short (целые числа, 2 байта)
- int (целые числа, 4 байта)
- long (целые числа, 8 байт)
- float (вещественные числа, 4 байта)
- double (вещественные числа, 8 байт)
- char (символ Unicode, 2 байта)
- boolean (значение true/false, 1 байт)

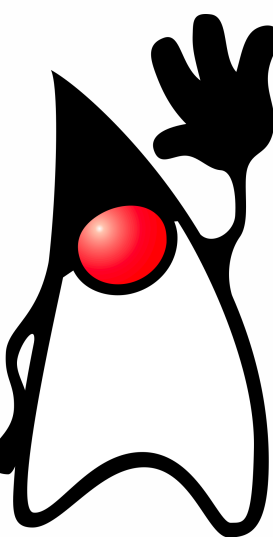


```
public static void main(String[] args){  
    byte a = 15;  
    int b = a;  
    System.out.println(b);  
}
```

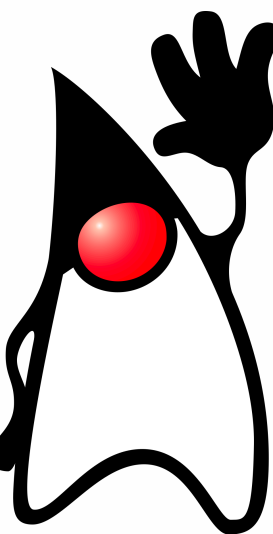
```
public static void main(String[] args){  
    int a=0;  
    long b=15;  
    a = (int) b;  
}
```

```
public static void main(String[] args){  
    double a=11.2345;  
    int b=(int)a;  
    System.out.println(b);  
}
```

```
public static void main(String[] args){  
    double a=128;  
    byte b=(byte)a;  
    System.out.println(b);  
}
```



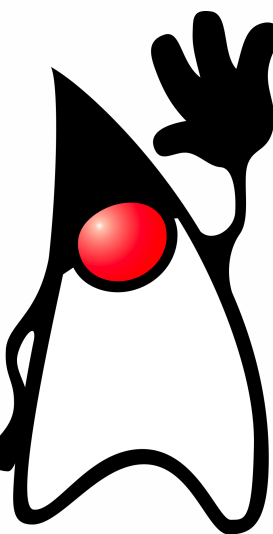
- Если Вы "кладёте" в большой контейнер содержимое меньшего контейнера», преобразование происходит автоматически, и ошибок возникать не должно.
- Если есть необходимость положить «значение из большего контейнера в меньший», нужно быть осторожным, и пользоваться явным приведением типов.
- При приведении float или double к целочисленным типам, дробная часть не округляется, а просто отбрасывается.
- Тип boolean не приводится ни к одному из типов.
- Тип char приводится к числовым типам, как код символа в системе UNICODE.
- Если число больше своего контейнера, результат будет непредсказуемым.



```
public static void main(String[] args){  
    double f = 0.0;  
    for (int i=1; i <= 10; i++) {  
        f += 0.1;  
    }  
}
```

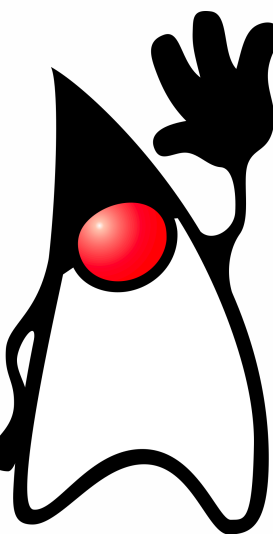
**Ответ: 0.999999999999999989**

**Для точной работы с десятичными дробями используйте BigDecimal**



Ссылочные (передаются по ссылке):

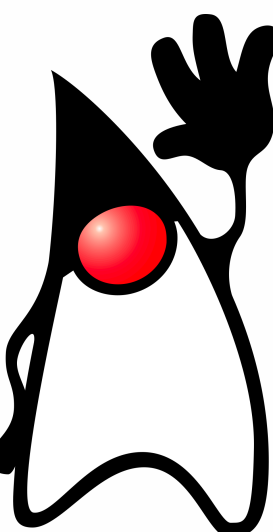
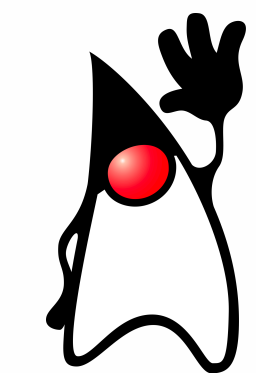
- String
- BigInteger
- BigDecimal
- Boolean
- Integer
- Long
- ...



# Ваши вопросы?

Если что — их можно задать  
ПОТОМ

- Quiz. Командная строка, ключевые команды
- Основные концепции программирования
- ООП в Java. Пакеты, классы, методы, параметры
- Типы данных в Java
- **Generic**
- Collection





05

# Generics

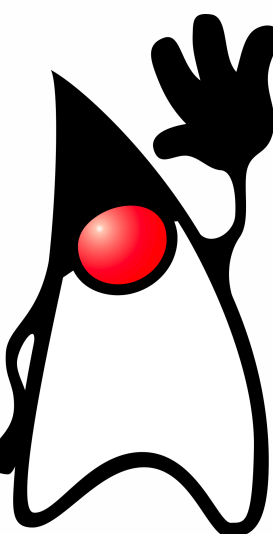
Зачем их придумали и как жили раньше?

Пример: Intro.java

Что дают Generics:

Строгая типизация.

Лучшая читаемость кода



Приняты такие сокращения:

E - Element

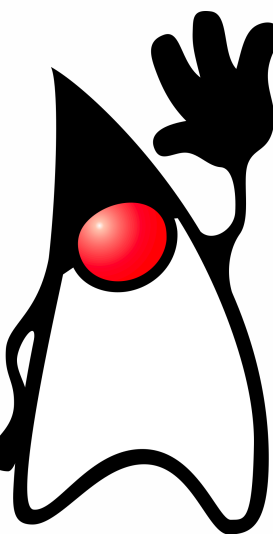
K - Key

N - Number

T - Type

V - Value

S,U,V etc. - 2nd, 3rd, 4th types

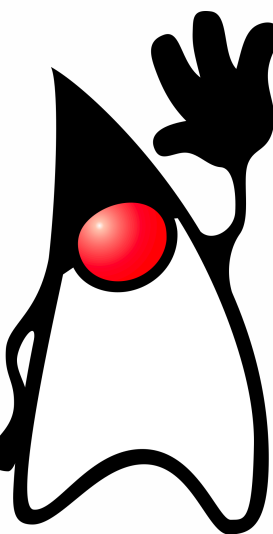


Generics классы

Generics методы

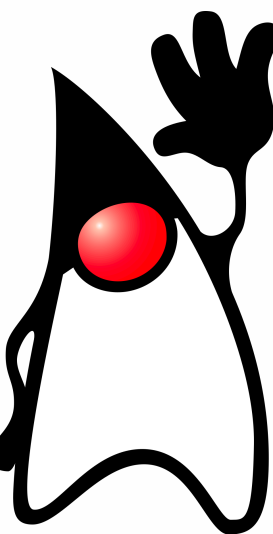
Пример: GenericsClass

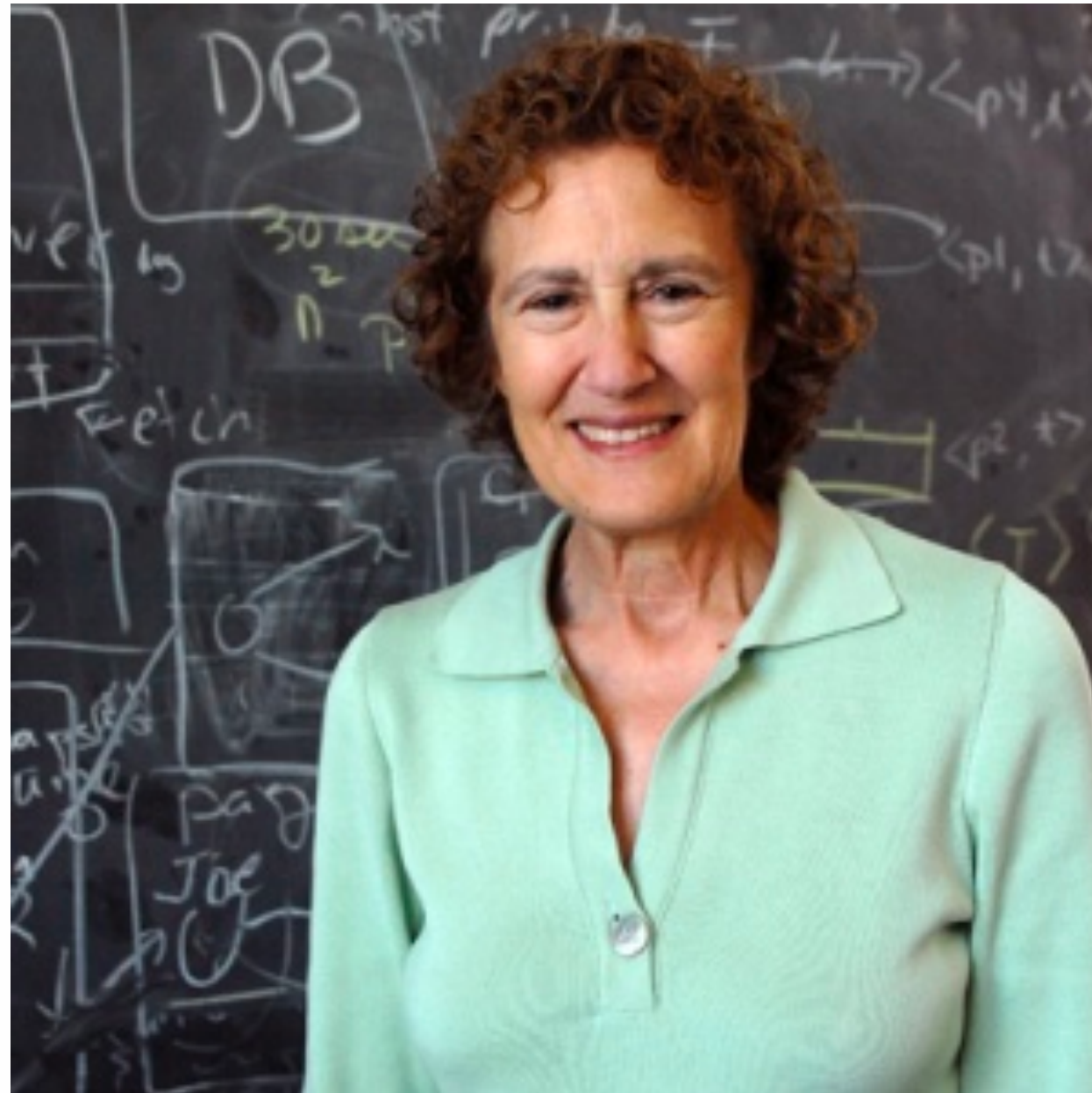
Пример: GenericsMethod



Пригодные для использования типы можно ограничить.

Пример: GenericsBounded

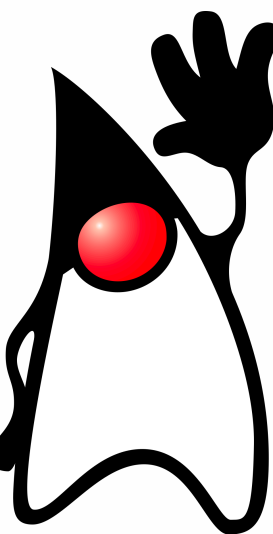




*Пусть  $q(x)$  является свойством верным относительно объектов  $x$  некоторого типа  $T$ . Тогда  $q(y)$  также должно быть верным для объектов  $y$  типа  $S$ , где  $S$  является подтипом типа  $T$ .*

*Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом.*

Роберт С. Мартин



Class Cat – наследник Class Animal,

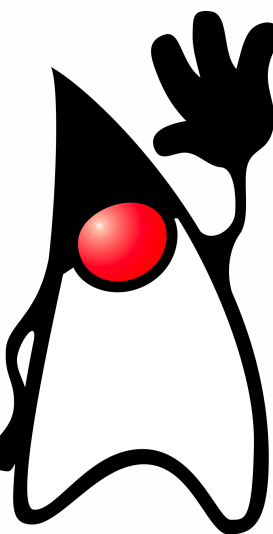
т.е. КОТ – ЭТО ЖИВОТНОЕ.

List<Cat> - это не наследник List<Animal>

т.е. группа котов – это не наследник группы животных

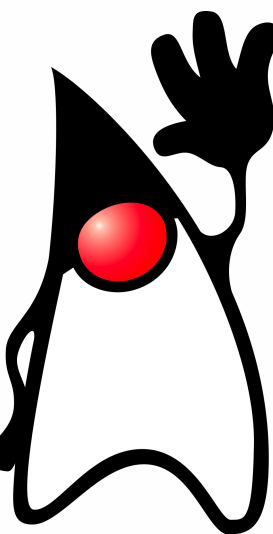
Наследие определено между индивидами,  
но не группами

Пример: GenericsInheritance



Если надо обработать коллекции родственных типов,  
используют WildCard – (?)  
(один из вариантов перевода – Джокер)

Пример WildCard 1,2





## Принцип PECS (Producer Extends Consumer Super)

Случай 1: Надо пройти по готовой, заполненной коллекции и что-то элементами

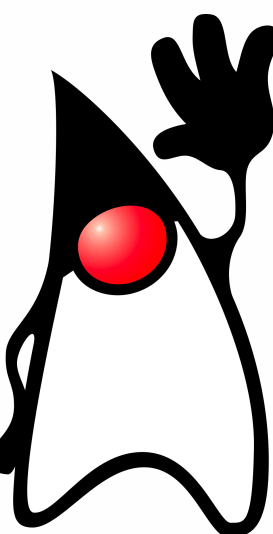
сделать с

Collection<? extends Cat>.

Случай 2: Коллекция изначально пустая, ее надо заполнить.

Collection<? super Cat>.

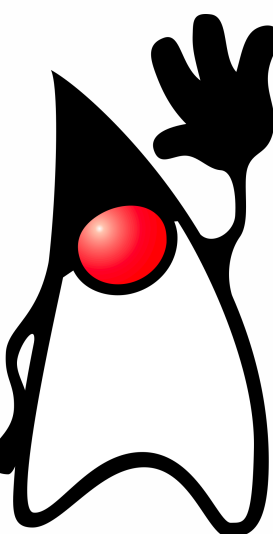
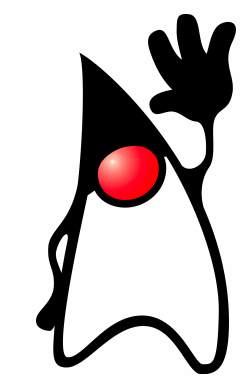
Пример: WildcardPECS



# Ваши вопросы?

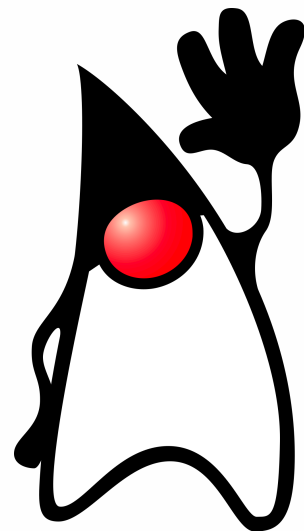
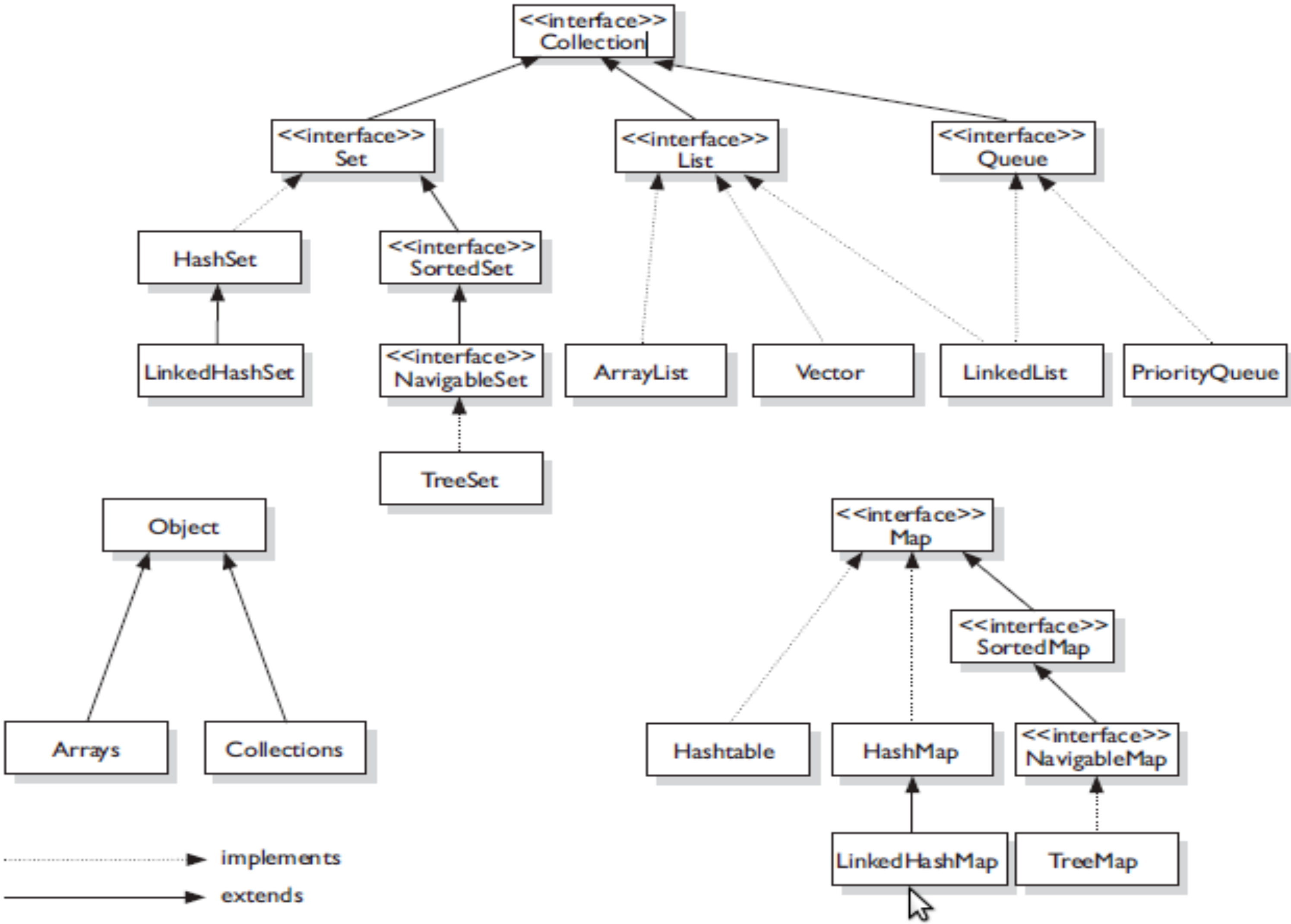
Если что — их можно задать  
ПОТОМ

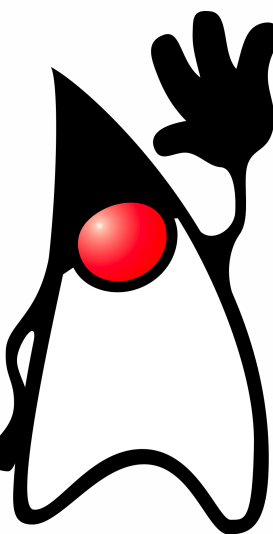
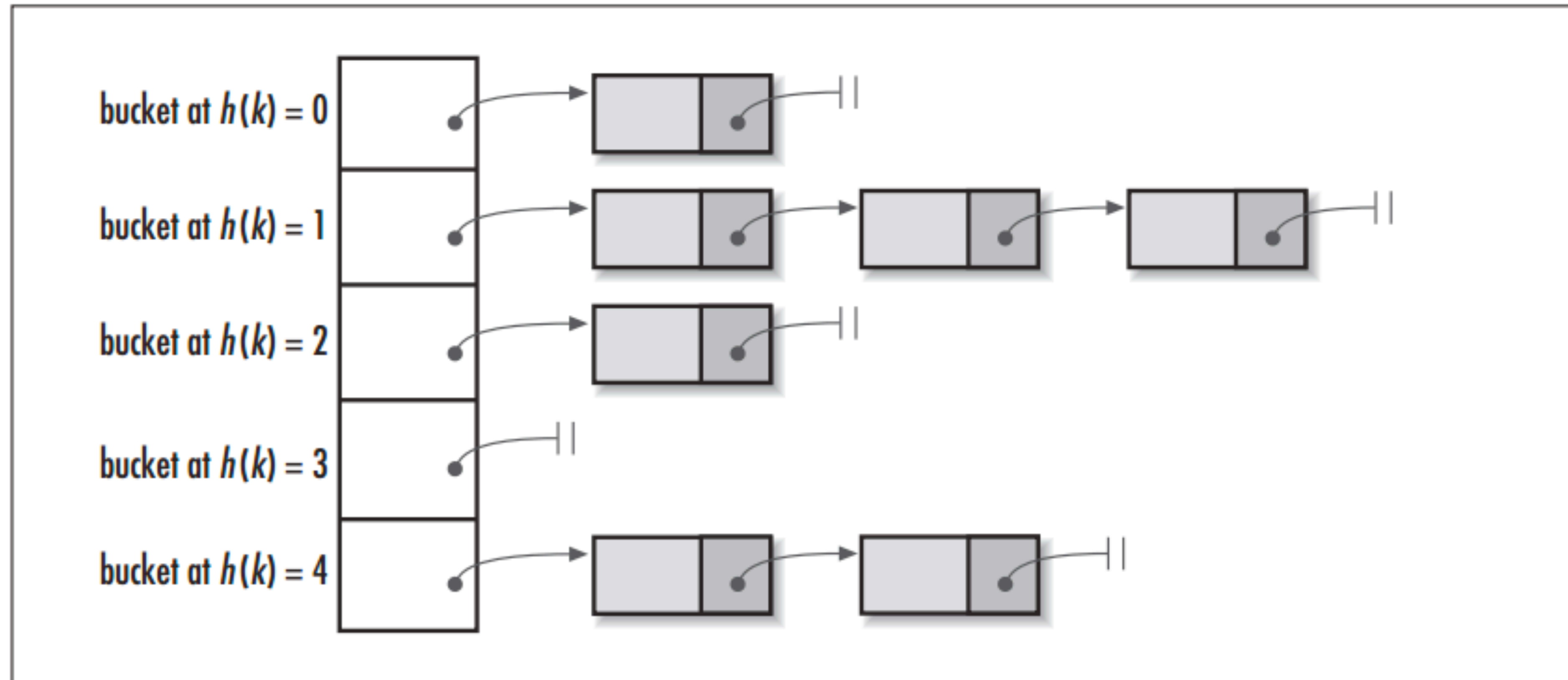
- Quiz. Командная строка, ключевые команды
- Основные концепции программирования
- ООП в Java. Пакеты, классы, методы, параметры
- Типы данных в Java
- Generic
- **Collections**



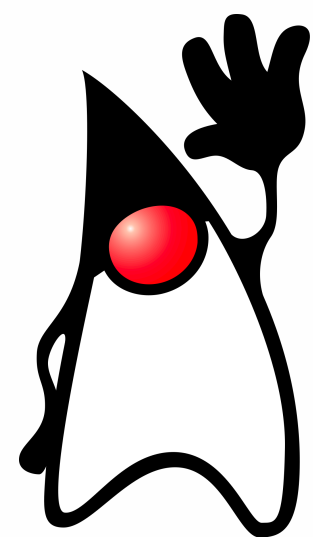
06

# Collections



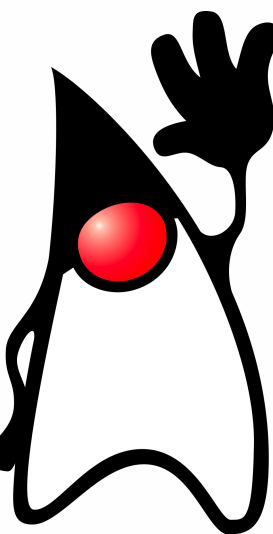


	Access by index	search	ordered	sorted
ArrayList	$O(1)$	$O(n)$	+	-
LinkedList	$O(n)$	$O(n)$	+	-
HashSet	$O(1)$	$O(n)$	-	-
TreeSet	$O(\log n)$	$O(\log n)$	+	+
HashMap	$O(1)$	$O(n)$	-	-



Неочевидные особенности

Пример: ArrayListDemo





# Ваши вопросы?

Если что — их можно задать  
ПОТОМ

07

ДЗ

Написать свою реализацию ArrayList на основе массива.

```
class DIYArrayList<T> implements List<T>{...}
```

Проверить, что на ней работают методы

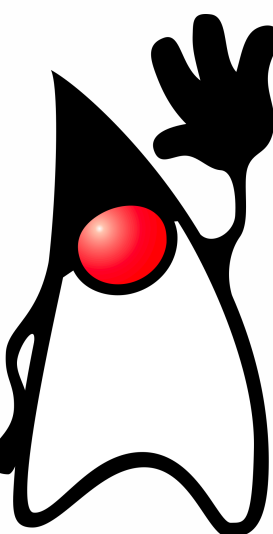
```
addAll(Collection<? super T> c, T... elements)
```

```
static <T> void copy(List<? super T> dest, List<? extends T> src)
```

```
static <T> void sort(List<T> list, Comparator<? super T> c)
```

из java.util.Collections

- 1) Проверяйте на коллекциях с 20 и больше элементами.
- 2) DIYArrayList должен имплементировать ТОЛЬКО ОДИН интерфейс - List.
- 3) Если метод не имплементирован, то он должен выбрасывать исключение UnsupportedOperationException.



**Ваши вопросы?**

**Спасибо за внимание!**

