

REFLECTION

- Библиотека, позволяющая оперировать информацией о типах во время выполнения
- Пакеты
 - `java.lang`
 - `java.lang.reflect`

ВВЕДЕНИЕ

ЧАСТЬ 1



ИНФОРМАЦИЯ О ТИПЕ

- Класс `Class<T>` -- информация о типе
- Предоставляемая информация
 - Структура класса
 - Структура наследования
 - Проверки времени выполнения
 - ...

ПОЛУЧЕНИЕ ИНФОРМАЦИИ О ТИПЕ

- Во время исполнения
`object.getClass()`
- Во время компиляции
`Type.class`
- Примеры
`ArrayList.class`
`int.class`
`int[].class`
- Предопределенные
`Wrapper.TYPE`
`Integer.TYPE`

ПРИМЕР

```
class Candy {
    static { System.out.println("Загрузка класса Candy"); }
}

class Gum {
    static { System.out.println("Загрузка класса Gum"); }
}

public class Example01 {
    public static void main(String[] args) {
        System.out.println("В методе main()");
        new Candy();
        System.out.println("После создания объекта Candy");
        try {
            Class.forName("Gum");
        } catch(ClassNotFoundException e) {
            System.out.println("Класс Gum не найден");
        }
        System.out.println("После вызова метода Class.forName(\"Gum\")");
    }
}
```

ПРИМЕР

```
class Candy {  
    static { System.out.println("Загрузка класса Candy"); }  
}
```

```
class Gum {  
    static { System.out.println("Загрузка класса Gum"); }  
}
```

```
public class Example01 {  
    public static void main(String[] args) {  
        System.out.println("в методе main()");  
        new Candy();  
        System.out.println("После создания объекта Candy");  
        try {  
            Class.forName("Gum");  
        } catch (ClassNotFoundException e) {  
            System.out.println("Класс Gum не найден");  
        }  
        System.out.println("После вызова метода Class.forName(\"Gum\")");  
        new Cookie();  
        System.out.println("После создания объекта Cookie");  
    }  
}
```

в методе main()
Загрузка класса Candy
После создания объекта Candy
Загрузка класса Gum
После вызова метода Class.forName("Gum")
Загрузка класса Cookie
После создания объекта Cookie

ПРИМЕР

```
class Initable {
    static final int staticFinal = 47;
    static final int staticFinal2 = Example02.rand.nextInt(1000);
    static { System.out.println("Initializing Initable"); }
}
class Initable2 {
    static int staticNonFinal = 147;
    static { System.out.println("Initializing Initable2"); }
}
class Initable3 {
    static int staticNonFinal = 74;
    static { System.out.println("Initializing Initable3"); }
}
public class Example02 {
    public static Random rand = new Random();
    public static void main(String[] args) throws Exception {
        Class<?> initable = Initable.class;
        System.out.println("1: " + "After creating Initable ref");
        System.out.println("2: " + Initable.staticFinal);
        System.out.println("3: " + Initable.staticFinal2);
        System.out.println("4: " + Initable2.staticNonFinal);
        Class<?> initable3 = Class.forName("Initable3");
        System.out.println("5: " + "After creating Initable3 ref");
        System.out.println("6: " + Initable3.staticNonFinal);
    }
}
```

ПРИМЕР

```
class Initable {
    static final int staticFinal = 47;
    static final int staticFinal2 = Example02.rand.nextInt(1000);
    static { System.out.println("Initializing Initable"); }
}
class Initable2 {
    static int staticNonFinal = 147;
    static { System.out.println("Initializing Initable2"); }
}
class Initable3 {
    static int staticNonFinal = 74;
    static { System.out.println("Initializing Initable3"); }
}
public class Example02 {
    public static Random rand = new Random();
    public static void main(String[] args) throws Exception {
        Class<?> initable = Initable.class;
        System.out.println("1: " + "After creating Initable");
        System.out.println("2: " + Initable.staticFinal);
        System.out.println("3: " + Initable.staticFinal2);
        System.out.println("4: " + Initable2.staticNonFinal);
        Class<?> initable3 = Class.forName("Initable3");
        System.out.println("5: " + "After creating Initable3");
        System.out.println("6: " + Initable3.staticNonFinal);
    }
}
```

```
1: After creating Initable ref
2: 47
3: 932
4: 147
5: After creating Initable3 ref
6: 74
```


ТИПЫ ТИПОВ

- Для определения типов служат методы объекта типа Class вида
 - `is*()`

Аннотация	<code>isAnnotation</code>
Массив	<code>isArray</code>
Примитивный	<code>isPrimitive</code>
Перечисление	<code>isEnum</code>
Интерфейс	<code>isInterface</code>
Класс	<code>is*Class</code>
Анонимный класс	<code>isAnonymousClass</code>
Локальный класс	<code>isLocalClass</code>
Класс-член	<code>isMemberClass</code>

ОБЩАЯ ИНФОРМАЦИЯ О КЛАССЕ

- Имя класса
 - `getCanonicalName()` – каноническое имя
 - `getName()` – полное имя
 - `getSimpleName()` – простое имя
- Структура классов
 - `getSuperClass()` – предок
 - `getInterfaces()` – реализуемые интерфейсы
- Модификаторы
 - `getModifiers()` – модификаторы

МЕСТО ОПРЕДЕЛЕНИЯ КЛАССА

- Методы получения места, в котором определен класс

Тип класса	Метод
Верхнего уровня	<code>getPackage()</code>
Вложенный	<code>getDeclaredClass()</code>
в конструктор	<code>getEnclosingConstructor()</code>
в метод	<code>getEnclosingMethod()</code>

ПРИВЕДЕНИЕ ТИПОВ

- Определение возможности приведения
 - `isAssignableFrom(class)` – класса
 - `isInstance(object)` – объекта
- Приведение
 - `cast(object)` – привести ссылку к типу
- Для объектов
 - `if (x instanceof String) {...}`

ПРИМЕР

```
class Building {}  
class House extends Building {}  
  
public class Example03 {  
    public static void main(String[] args) {  
        Building b = new House();  
        Class<House> houseType = House.class;  
        House h = houseType.cast(b);  
        h = (House)b; // ... or just do this.  
    }  
}
```

ПРИМЕР

```
class Base {}
class Derived extends Base {}
public class Example04 {
    static void test(Object x) {
        System.out.println("Testing x of type " + x.getClass());
        System.out.println("x instanceof Base " + (x instanceof Base));
        System.out.println("x instanceof Derived "+ (x instanceof Derived));
        System.out.println("Base.isInstance(x) "+ Base.class.isInstance(x));
        System.out.println("Derived.isInstance(x) " +
Derived.class.isInstance(x));
        System.out.println("x.getClass() == Base.class " + (x.getClass() ==
Base.class));
        System.out.println("x.getClass() == Derived.class " + (x.getClass() ==
Derived.class));
    }
    public static void main(String[] args) {
        test(new Base()); test(new Derived());
    }
}
```

ПРИМЕР

```
class Base {}
class Derived extends Base {}
public class Example04 {
    static void test(Object x) {
        System.out.println("Testing x of type class Base");
        System.out.println("x instanceof Base true");
        System.out.println("x instanceof Derived false");
        System.out.println("Base.isInstance(x) true");
        System.out.println("Derived.isInstance(x) false");
        System.out.println("x.getClass() == Base.class true");
        System.out.println("x.getClass() == Derived.class false");
        System.out.println("x.getClass().equals(Base.class) true");
        System.out.println("x.getClass().equals(Derived.class) false");
        System.out.println("-----");
        System.out.println("Testing x of type class Derived");
        System.out.println("x instanceof Base true");
        System.out.println("x instanceof Derived true");
        System.out.println("Base.isInstance(x) true");
        System.out.println("Derived.isInstance(x) true");
        System.out.println("x.getClass() == Base.class false");
        System.out.println("x.getClass() == Derived.class true");
        System.out.println("x.getClass().equals(Base.class) false");
        System.out.println("x.getClass().equals(Derived.class) true");
    }
    public static void main(String[] args) {
        test(new Base()); test(new Derived());
    }
}
```

```
Testing x of type class Base
x instanceof Base true
x instanceof Derived false
Base.isInstance(x) true
Derived.isInstance(x) false
x.getClass() == Base.class true
x.getClass() == Derived.class false
x.getClass().equals(Base.class) true
x.getClass().equals(Derived.class) false
-----
Testing x of type class Derived
x instanceof Base true
x instanceof Derived true
Base.isInstance(x) true
Derived.isInstance(x) true
x.getClass() == Base.class false
x.getClass() == Derived.class true
x.getClass().equals(Base.class) false
x.getClass().equals(Derived.class) true
```

СТРУКТУРА КЛАССА

ЧАСТЬ 2



ЧТО БЫВАЕТ ВНУТРИ КЛАССА

- **Members** - Просто какие-то члены
- **Field** – поля
- **Method** – методы
- **Constructor** – конструкторы
- **Class** – вложенные классы

ИНФОРМАЦИЯ О ЧЛЕНЕ КЛАССА

- Интерфейс `Member`
- Методы
 - `getDeclaringClass()` – класс, в котором определен
 - `getName()` – имя члена
 - `getModifiers()` – модификаторы

МОДИФИКАТОРЫ

- Класс `Modifiers`

Константа	Метод	Модификатор
ABSTRACT	<code>isAbstract</code>	<code>abstract</code>
FINAL	<code>isFinal</code>	<code>final</code>
INTERFACE	<code>isInterface</code>	<code>interface</code>
NATIVE	<code>isNative</code>	<code>native</code>
PRIVATE	<code>isPrivate</code>	<code>private</code>
PROTECTED	<code>isProtected</code>	<code>protected</code>
PUBLIC	<code>isPublic</code>	<code>public</code>
STATIC	<code>isStatic</code>	<code>static</code>
STRICT	<code>isStrict</code>	<code>strictfp</code>
SYNCHRONIZED	<code>isSynchronized</code>	<code>synchronized</code>
TRANSIENT	<code>isTransient</code>	<code>transient</code>
VOLATILE	<code>isVolatile</code>	<code>volatile</code>

ПОЛЯ

- Открытые
 - `getFields()` – все поля
 - `getField(name)` – конкретное поле
- Все
 - `getDeclaredFields()` – все поля
 - `getDeclaredField(name)` – конкретное поле
- Исключения
 - `NoSuchFieldException`

СВОЙСТВА ПОЛЕЙ

- Класс `Field`
- Информация
 - `getName()` – имя поля
 - `getType()` – тип значения
- Чтение значения
 - `get(object)` – ссылки
 - `get*(object)` – значения примитивного типа
- Запись значения
 - `set(object, value)` – ссылки
 - `set*(object, value)` – значения примитивного типа

МЕТОДЫ

- Открытые
 - `getMethods()` – все методы
 - `getMethod(name, Class... parameters)` – конкретный метод
- Все
 - `getDeclaredMethods()` – все методы
 - `getDeclaredMethod(name, Class... parameters)` – конкретный метод
- Исключения
 - `NoSuchMethodException`

СВОЙСТВА МЕТОДОВ

- Класс `Method`
- Сигнатура метода
 - `getName()` – имя метода
 - `getParameterTypes()` – параметры метода
- Другая информация
 - `getExceptionTypes()` – возможные исключения
 - `getReturnType()` – тип возвращаемого значения
- Вызов метода
 - `invoke(Object object, Object ...args)` – вызвать метод с указанными аргументами
 - `InvocationTargetException` – в случае ошибки

КОНСТРУКТОРЫ

- Открытые
 - `getConstructors()` – все конструкторы
 - `getConstructor(Class... parameters)` – конкретный конструктор
- Все
 - `getDeclaredConstructors()` – все конструкторы
 - `getDeclaredConstructor(Class... parameters)` – конкретный конструктор
- Исключения
 - `NoSuchMethodException`

СВОЙСТВА КОНСТРУКТОРОВ

- Класс `Constructor`
- Информация о конструкторе
 - `getParameterTypes()` – параметры конструктора
 - `getExceptionTypes()` – возможные исключения
- Создание объекта
 - `newInstance(Object ... args)` – создать новый объект
 - `class.newInstance()` – создать новый объект используя конструктор по умолчанию

КЛАССЫ И ИНТЕРФЕЙСЫ

- Открытые
 - `getClasses()` – все классы и интерфейсы
- Все
 - `getDeclaredClasses()` – все классы и интерфейсы

ДОСТУП К ЗАКРЫТЫМ ЧЛЕНАМ

- По умолчанию доступ к закрытым членам запрещен → `IllegalAccessException`
- Все члены `extends AccessibleObject`
 - `setAccessible(boolean)` – запросить доступ
 - `isAccessible()` – проверить доступ

ПРИМЕР: ЛИСТИНГ КЛАССА

```
Class c = ...;  
for (Field m : c.getDeclaredFields()) {  
    System.out.println(m);  
}  
  
for (Constructor m : c.getDeclaredConstructors()) {  
    System.out.println(m);  
}  
  
for (Method m : c.getDeclaredMethods()) {  
    System.out.println(m);  
}
```

ПРИМЕР: СОЗДАНИЕ ЭКЗЕМПЛЯРА

```
// Получение класса
Class<Integer> clazz = Integer.class;
// Получение конструктора
Constructor<Integer> c = clazz.getConstructor(int.class);
// Создание экземпляра
Integer i = (Integer) c.newInstance(100);
// Проверка
System.out.println(i);
```

ПРИМЕР

```
class PrivateData {
    private int priv = 1;
    public int pub;
    public PrivateData(int a) {
        System.out.println("Hello!");
        this.priv = a;
        this.pub = a;
    }
    private void foo() { System.out.println("Hidden! " + priv); }
    public void boo(int num) { System.out.println("Public " + num); }
}
class PrivateData2 extends PrivateData {
    private int priv2 = 0;
    public int pub2 = 1;
    public PrivateData2(int a) {
        super(a);
    }
}
```

ПРИМЕР

```
public static void printModifiers(Class<?> c) {  
    int mods = c.getModifiers();  
    if (Modifier.isPublic(mods)) { System.out.print("public "); }  
    if (Modifier.isAbstract(mods)) { System.out.print("abstract "); }  
    if (Modifier.isFinal(mods)) { System.out.print("final "); }  
}
```

```
static void printSuperClasses(Class<?> c, String tabs) {  
    if (c == null) { return; }  
    printSuperClasses(c.getSuperclass(), tabs + "  ");  
    System.out.print(tabs);  
    printModifiers(c);  
    System.out.println(c.getName());  
}
```

ПРИМЕР

```
public static void printFields(Class<?> c) {
    Field[] publicFields = c.getFields();
    for (Field field : publicFields) {
        Class<?> fieldType = field.getType();
        System.out.print("Имя: " + field.getName() + " ");
        System.out.println("Тип: " + fieldType.getName());
    }
}

public static void printAllFields(Class<?> c) {
    Field[] declaredFields = c.getDeclaredFields();
    for (Field field : declaredFields) {
        Class<?> fieldType = field.getType();
        System.out.print("Имя: " + field.getName() + " ");
        System.out.println("Тип: " + fieldType.getName());
    }
}
```

```
Имя: pub2 Тип: int
Имя: pub Тип: int
-----
Имя: priv2 Тип: int
Имя: pub2 Тип: int
```


ПРИМЕР

```
PrivateData pd = new PrivateData(5);  
Field field = null;  
Integer value = 0;  
field = PrivateData.class.getField("pub");  
value = (Integer) field.get(pd);  
System.out.println(value);
```

```
field = PrivateData.class.getDeclaredField("priv");  
field.setAccessible(true);  
value = (Integer) field.get(pd);  
field.setInt(pd, value + 1);  
value = (Integer) field.get(pd);  
System.out.println(value);
```

ПРИМЕР

```
Class<? extends PrivateData> pdClass = pd.getClass();  
Class<?>[] paramTypes = new Class[] { int.class };  
Method method = null;  
method = pdClass.getMethod("boo", paramTypes);  
Object[] params = new Object[] { new Integer(10) };  
method.invoke(pd, params);
```

ПРИМЕР

```
Class<?>[] paramTypes = new Class[] { int.class };
Constructor<?> aConstrct =
pdClass.getConstructor(paramTypes);
Object[] params = new Object[] { 5 };
aConstrct.newInstance(params);

Class<?> t = Class.forName("java.lang.String");
Object obj = t.newInstance();
```

МАССИВЫ

ЧАСТЬ 3



ОПЕРАЦИИ С МАССИВАМИ

- Класс `Array`
- Создание массива заданного типа
 - `newInstance(Class, length)` – линейного
 - `newInstance(Class, dims[])` – многомерного
- Чтение значения из массива
 - `get(array, index)` – ссылки
 - `get*(array, index)` – значения примитивного типа
- Запись значения в массив
 - `set(array, index, value)` – ссылки
 - `set*(array, index, value)` – значения примитивного типа

МАССИВЫ КАК ТИПЫ

- Методы
 - `isArray()` – является ли массивом
 - `getComponentType()` – тип элемента массива

ЗАГРУЗЧИКИ КЛАССОВ

ЧАСТЬ 4

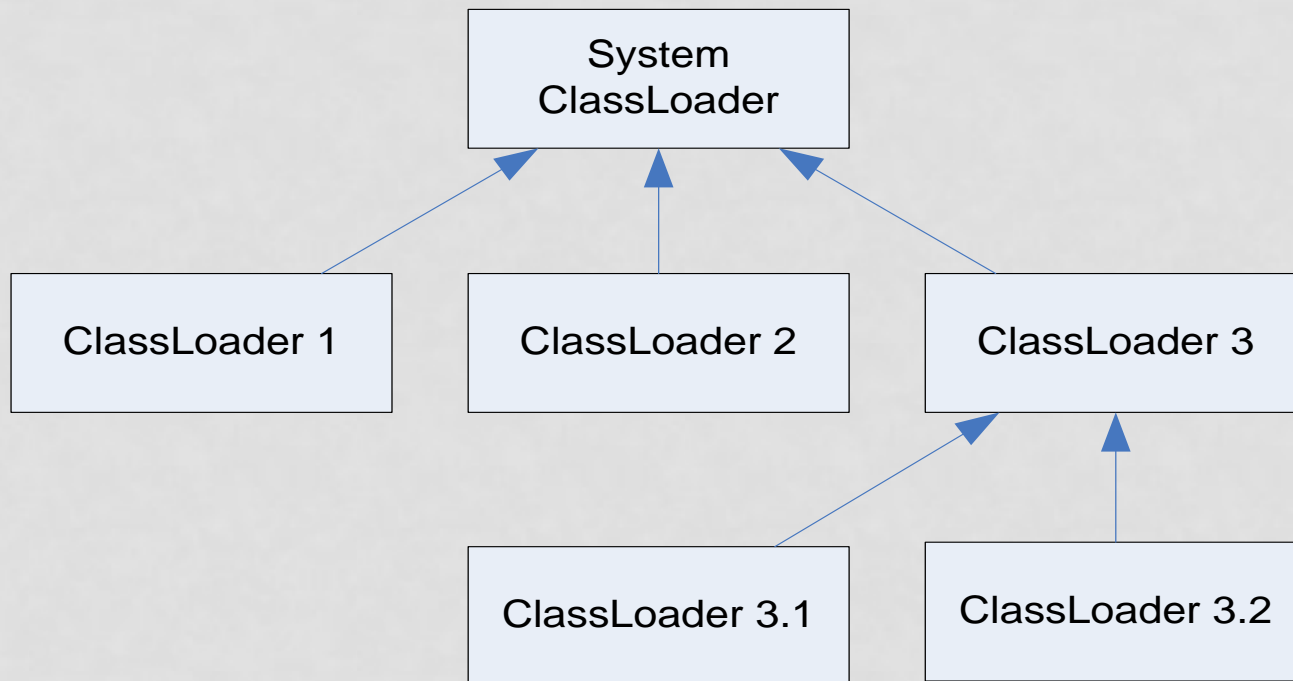


ЗАГРУЗЧИКИ КЛАССОВ

- Позволяют загружать и определять новые классы
- Класс `ClassLoader`
- Методы
 - `loadClass(name, resolve?)` – загружает класс по имени
 - `findLoadedClass(name)` – найти уже загруженный класс
 - `resolveClass(class)` – загружает библиотеки

ДЕРЕВО ЗАГРУЗЧИКОВ

- Загрузчики образуют дерево
- Загрузчики в разных ветвях могут загрузить разные классы с одним полным именем



ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

- Получения родителя
 - `getParent()`
- Загрузка ресурсов
 - `URL getResource(String name)` – определение местоположения ресурса по имени
 - `getResourceAsStream(String name)` – чтение ресурса по имени

ЗАГРУЗЧИКИ И КЛАССЫ

- Получение загрузчика
 - `getClassLoader()` – кто загрузил класс
 - `Thread.getContextClassLoader()` – контекстный загрузчик
- “Прямая” загрузка класса
 - `Class.forName(name)`
 - `Class.forName(name, init, ClassLoader)`

РЕАЛИЗАЦИИ ЗАГРУЗЧИКОВ

- Класс `URLClassLoader`
 - Загружает классы из нескольких мест, заданных `URL`

ПРИМЕР: ЗАГРУЗКА КЛАССА

```
URL jar = new URL("file:///");  
className = "Test";  
ClassLoader cl = new URLClassLoader(new URL[]{jar});  
Class c = cl.loadClass(className);  
  
Method m = c.getMethod("main", String[].class);  
m.invoke(null, (Object) new String[]{"hello"});
```

ПАРАМЕТРЫ ТИПОВ

ЧАСТЬ 5



ИНФОРМАЦИЯ О ПАРАМЕТРАХ ТИПОВ (ПТ)

- Информация о конкретных параметрах типов стирается
- Информация о зависимостях типов сохраняется

ПОЛУЧЕНИЕ ИНФОРМАЦИЯ О ПТ

- Для классов
 - `getGenericSuperclass()`
 - `getGenericInterfaces()`
- Для методов и конструкторов
 - `getGenericParameterTypes()`
 - `getGenericReturnType()`
 - `getGenericExceptionTypes()`
- Для полей
 - `getGenericType()`

ПРЕДСТАВЛЕНИЕ ИНФОРМАЦИИ О ПТ

- Интерфейс `Type`

- Классы
- Параметризованный класс
- Переменная типа
- Wildcard
- Массивы

`Class`

`ParameterizedType`

`TypeVariable`

`WildcardType`

`GenericArrayType`

ПАРАМЕТРИЗОВАННЫЕ КЛАССЫ

- Пример: `Collection<String>`
- Интерфейс `ParameterizedType`
 - `getRawType()` – не параметризованный тип
 - `getActualTypeArguments()` – реальные аргументы типа

ПЕРЕМЕННЫЕ ТИПА

- Пример: T
- Получение
 - `getTypeParameters()`
- Интерфейс `TypeVariable`
 - `getName()` – имя переменной
 - `getBounds()` – верхние границы
 - `getGenericDeclaration()` – кто объявил

WILDCARDS

- Пример: ? super HashSet extends Collection
- Интерфейс Wildcard
 - `getUpperBounds()` – Верхние границы
 - `getLowerBounds()` – Нижние границы

МАССИВЫ

- Тип элемента – переменная типа
 - Пример: `T[]`
- Тип элемента – параметризованный тип
 - Пример: `Set<T>[]`
- Интерфейс `GenericArrayType`
 - `getGenericComponentType()` – тип элемента

ВЫВОДЫ

- Reflection позволяет
 - Анализировать классы по время исполнения
 - Загружать классы по имени
 - Создавать экземпляры классов по имени
 - Вызывать метод классов по имени
 - Оперировать значениями полей по имени
 - Создавать и оперировать с массивами по типу элемента