СБЕРБАНК ТЕХНОЛОГИИ

Reflection

УЗНАЕМ

СБЕРБАНК ТЕХНОЛОГИИ

- Какую метаинформацию можно получить в рантайме о классах?
- Можно ли звать приватные методы класса из других классов?
- Зачем и как это делать ?

КЛАСС JAVA.LANG.CLASS

СБЕРБАНК ТЕХНОЛОГИИ

Содержит методы для получения полной информации о классе, вызова методов и изменения полей.

ДОСТУПНАЯ ИНФОРМАЦИЯ

СБЕРБАНК ТЕХНОЛОГИИ

- Имя, пакет класса
- Методы: (список, тип возвращаемого значения, имена и типы аргументов, модификаторы видимости)
- Поля (список, имена, типы, модификаторы видимости)
- Иерархия класса
- Возможность вызвать методы, изменить поля

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

- Получить список всех полей, проверить, что их значения != null
- *Склонировать объект*
- Скопировать состояние объекта в другой
- Кешировать результаты вызова методов(совместно с Proxy)

КАК ПОЛУЧИТЬ ОБЪЕКТ CLASS

СБЕРБАНК ТЕХНОЛОГИИ

```
Class<Integer> c =  Integer.class;

Class<String> c =  String.class;

someObject.getClass();
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Список всех public методов, объявленных в классе или унаследованных
public Method[] getMethods()


//Список всех методов, объявленных в классе
public Method[] getDeclaredMethods()
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Метод с заданным именем и аргументами
 public Method getMethod (String name,
                          Class<?>...parameterTypes)


 Method m = String.class.getMethod("replaceAll",
                                   String.class, String.class)
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Список всех public полей, объявленных в классе или унаследованных
public Field[] getFields()

//Список всех полей, объявленных в классе
public Field[] getDeclaredFields()

//поле по имени
public Field getField(String name)
```

ВАЖНЫЕ МЕТОДЫ КЛАССА

СБЕРБАНК ТЕХНОЛОГИИ

```
//Возвращает класс родителя
public native Class<? super T> getSuperclass();
```

ЧТО БУДЕТ НА КОНСОЛИ?

СБЕРБАНК ТЕХНОЛОГИИ

```
System.out.println(String.class.getSuperclass());

System.out.println(Object.class.getSuperclass());
```

У КЛАССА OBJECT SUPERCLASS == NULL

СБЕРБАНК ТЕХНОЛОГИИ

```
//Object
System.out.println(String.class.getSuperclass());

//null
System.out.println(Object.class.getSuperclass());
```

ПОЛУЧЕНИЕ ПОЛНОЙ ИЕРАРХИИ

СБЕРБАНК ТЕХНОЛОГИИ

```
public static void printHierarchy(Class<?> clazz) {
    while (clazz != null) {
        System.out.println(clazz);
        clazz = clazz.getSuperclass();
    }
}
```

ИСПОЛЬЗОВАНИЕ REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

В слайдах содержатся простые для понимания примеры, которые показывают **ЧТО** можно делать через reflection.

Используйте рефлекшен, только если без него не обойтись.

Реальные полезные примеры ниже.

КАК СОЗДАТЬ ЭКЗЕМПЛЯР КЛАССА ?

СБЕРБАНК ТЕХНОЛОГИИ

```
try {
    //Зовется конструктор без параметров
    Person p = Person.class.newInstance();
} catch(InstantiationException | IllegalAccessException e){
  …
}
```

КАК СОЗДАТЬ ЭКЗЕМПЛЯР КЛАССА ?

СБЕРБАНК ТЕХНОЛОГИИ

```
// Зовется конструктор со String аргументом
Person p2 = Person.class.getConstructor(String.class)
                        .newInstance("Alex");
```

ВЫЗОВ МЕТОДА

СБЕРБАНК ТЕХНОЛОГИИ

```
private void setName(Object o, String name)
    throws NoSuchMethodException, InvocationTargetException, IllegalAccessException {

  Class<?> clazz = o.getClass();
  Method m = clazz.getMethod("setName", String.class);


  //мы передаем объект, у которого вызовется метод,
                                    и параметры метода
  m.invoke(o, name);

}
```

ВЫЗОВ ПРИВАТНОГО МЕТОДА

СБЕРБАНК ТЕХНОЛОГИИ

```
Method m = clazz.getMethod("setName", String.class);
m.setAccessible(true);
m.invoke(o, name);
```

ИЗМЕНЕНИЕ FINAL ПОЛЕЙ

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Person {
    private final String name;//Можно поменять?

    public Person(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

МОЖНО!

СБЕРБАНК ТЕХНОЛОГИИ

```
Person person = get();

Field name = Person.class.getField("name");
name.setAccessible(true);
name.set(person, "Julia");
```

ДЖЕНЕРИКИ ЧЕРЕЗ REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

Можно достать метаинформацию о дженериках на уровне **класса**.

Информация, чем параметризованны **локальные объекты** стирается.

ДЖЕНЕРИКИ, ДОСТУПНЫЕ ЧЕРЕЗ REFLECTION

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Runtime<T extends Number>
                          implements Callable<Double> {
    private final List<Integer> integers = emptyList();

    public List<T> numbers() {return emptyList();}

    public List<String> strings() {return emptyList();}

    @Override
    public Double call() {return 0d;}
}
```

@АННОТАЦИИ

СБЕРБАНК ТЕХНОЛОГИИ

Позволяют добавлять метаинформацию в класс.

Использовать эту информацию можно для разных целей.

@DEPRECATED

СБЕРБАНК ТЕХНОЛОГИИ

Помечаются устаревшие методы, нерекомендованные к использованию в новом коде

Пример из класс Date:

```
* @deprecated As of JDK version 1.1,
 * replaced by Calendar.get(Calendar.HOUR_OF_DAY).
 */
@Deprecated
public int getHours() {
    return normalize().getHours();
}
```

@OVERRIDE

СБЕРБАНК ТЕХНОЛОГИИ

Показывает что текущий метод переопределяет метод родителя или реализует интерфейс.
Компилятор проверяет, что помеченный метод действительно это делает.

```
@Override
public int run() {
    …

}
```

ПРИМЕР СВОЕЙ АННОТАЦИИ

СБЕРБАНК ТЕХНОЛОГИИ

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface NotNull {
}
```

@TARGET

СБЕРБАНК ТЕХНОЛОГИИ

Показывает на что можно вешать данную аннотацию

```
public enum ElementType {
    TYPE,
    FIELD,
    METHOD,
    PARAMETER,
    CONSTRUCTOR,
    LOCAL_VARIABLE,
    ANNOTATION_TYPE,
    PACKAGE,
    TYPE_PARAMETER,
    TYPE_USE
}
```

@RETENTION

СБЕРБАНК ТЕХНОЛОГИИ

Показывает на каком уровне доступна аннотация

```
public enum RetentionPolicy {
    SOURCE,
    CLASS, // по умолчанию
    RUNTIME
}
```

Возвращаемые значения могут быть примитивами, String, Class, enums

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface ValidLength {
    int min();

    int max();
}
```

Можно задавать default значения

```
@Target(ElementType.FIELD)
@Retention(RetentionPolicy.RUNTIME)
public @interface MinLength {
    int  value() default 3;
}
```

ЗАДАНИЕ АТРИБУТОВ

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Person {
    @ValidLength(min = 4, max = 10)
    private final String name;
    …
}
```

Название атрибута можно не указывать, если оно называется value

```
public class Person {
    @MinLength(3)
    private final String name;
    …
}
```

ПОЛУЧЕНИЕ ИНФОРМАЦИИ ОБ АННОТАЦИЯХ

СБЕРБАНК ТЕХНОЛОГИИ

```
Field f = …

if (f.isAnnotationPresent(ValidLength.class)) {
    ValidLength an=f.getAnnotation(ValidLength.class);
    int max = an.max();
    int min = an.min();
    …
}
```

ПРИМЕР ПРОВЕРКИ ПОЛЕЙ ПО АННОТАЦИЯМ

СБЕРБАНК ТЕХНОЛОГИИ

```
public void validateStringLength(Object o) throws Exception {
    Class<?> clazz = o.getClass();
    for (Field field : clazz.getDeclaredFields()) {
        if (field.isAnnotationPresent(ValidLength.class)) {
          ValidLength an= field.getAnnotation(ValidLength.class);
            int max = an.max();
            int min = an.min();

            String value = field.get(o).toString();
            if (value.length() < min) {
               throw new IllegalStateException(field.getName()
               + " length should be between " + min + " and " + max);
            }
        }
    }
}
```

DYNAMIC PROXY

СБЕРБАНК ТЕХНОЛОГИИ

Позволяет перехватывать в рантайме вызовы методов интерфейса и обрабатывать их.

Прокси может притворяться любым интерфейсом.

ПРИМЕР: КЕШИРУЮЩИЙ PROXY

СБЕРБАНК ТЕХНОЛОГИИ

Кеширующий прокси перехватывает вызовы интерфейса.

Если метод помечен аннотацией @Cache, то:
Проверяет есть ли в кеше результат, если есть, то возвращает его.
Иначе, вызывает реальный метод, кеширует результат и возвращает его.

Если метод не помечен аннотацией @Cache, просто делегирует метод реализации

ПРИМЕР CACHE PROXY

СБЕРБАНК ТЕХНОЛОГИИ

```
Calculator calculator = new CalculatorImpl();
calculator.calc(1);
calculator.calc(1); // повторный расчет


Calculator cached = ProxyUtils.makeCached(calculator);
cached.calc(1);
cached.calc(2);
cached.calc(1); // результат из кеша
```

ПРИМЕР: RMI ПРОКСИ

СБЕРБАНК ТЕХНОЛОГИИ

Прокси перехватывает вызовы интерфейса и перенаправляет их по сети другому серверу и возвращает результат.

```
Calculator calc = ProxyUtils.client(Calculator.class);
calc.calc(1); // перехват вызова и отправка удаленной
                                                машине

Service service = ProxyUtils.client(Service.class);
service.run();
```

ПРИМЕР: RMI ПРОКСИ

СБЕРБАНК ТЕХНОЛОГИИ

Это позволяет быстро создать клиент любого интерфейса(На удаленной машине должны быть слушатели вызова, созданные, например, тоже через Proxy).

```
Calculator calc = ProxyUtils.client(Calculator.class);
calc.calc(1); // перехват вызова и отправка удаленной
                                                машине

Service service = ProxyUtils.client(Service.class);
service.run();
```

КАК СОЗДАТЬ ПРОКСИ. java.lang.reflect.Proxy

СБЕРБАНК ТЕХНОЛОГИИ

```
public class Proxy {

//возвращает объект, который реализует интерфейсы interfaces[]
//вызов методов передается в реализацию InvocationHandler
public static Object newProxyInstance(ClassLoader loader,
                                       Class<?>[] interfaces,
                                       InvocationHandler h)
}
```

INVOCATION HANDLER

СБЕРБАНК ТЕХНОЛОГИИ

Поведение прокси задается в реализации интерфейса InvocationHandler

```
public interface InvocationHandler {
    Object invoke(Object proxy, Method method,
                  Object[] args)  throws Throwable;
}
```

ПРИМЕР LOG ХЕНДЛЕРА

СБЕРБАНК ТЕХНОЛОГИИ

```
public class LogHandler implements InvocationHandler {
    private final Object delegate;

    public LogHandler(Object delegate) {
        this.delegate = delegate;
    }

    @Override
    public Object invoke(Object proxy, Method method,
                         Object[]args) throws Throwable {
        System.out.println("Started " + method.getName());
        Object result = method.invoke(delegate, args);
        System.out.println("Finished " + method.getName() + ".
                                    Result " + result);
        return result;
    }
```

ДОБАВЛЯЕМ ЛОГИРОВАНИЕ ВСЕХ МЕТОДОВ ЛИСТА

СБЕРБАНК ТЕХНОЛОГИИ

```
List<String> list = new ArrayList<>();

List<String> loggedList = (List<String>)
            Proxy.newProxyInstance(
                  ClassLoader.getSystemClassLoader(),
                  list.getClass().getInterfaces(),
                  new LogHandler(list)
            );
```

ЗАДАНИЯ

СБЕРБАНК ТЕХНОЛОГИИ

Вывести на консоль все методы класса, включая все родительские методы (включая приватные)

Вывести все геттеры класса

Проверить что все String константы имеют значение = их имени

```
public static final String MONDAY = "MONDAY";
```

Реализовать кэширующий прокси

ДЗ

СБЕРБАНК ТЕХНОЛОГИИ

Просмотреть основные моменты работы с reflection и dynamic proxy: http://tutorials.jenkov.com/java-reflection/index.html

Реализовать следующий класс по документации

```
public class BeanUtils {
   /**
    * Scans object "from" for all getters. If object "to"
    * contains correspondent setter, it will invoke it
    * to set property value for "to" which equals to the property
    * of "from".
    * <p/>
    * The type in setter should be compatible to the value returned
    * by getter (if not, no invocation performed).
    * Compatible means that parameter type in setter should
    * be the same or be superclass of the return type of the getter.
    * <p/>
    * The method takes care only about public methods.
    *
    * @param to   Object which properties will be set.
    * @param from Object which properties will be used to get values.
    */
   public static void assign(Object to, Object from) {... }
}
```