Семинар №3

Задачи семинара

1. Вопросы и обсуждение семинара №2
2. Класс java.lang.Class. Получение объекта класса Class: свойство .class, Object.getClass(), Class.forName(). Доступ до ресурсных файлов.
3. Динамическое создание объекта класса через Class.newInstance()
4. Классы Field, Contructor, Method (AccessibleObject). Динамический вызов методов объекта через Method.invoke(). Метод AccessibleObject.setAccessible(boolean flag)
5. Класс java.lang.ClassLoader. Возможность динамического создания класса. Иерархия class loaders.
6. Класс java.lang.reflect.Proxy
7. Использование аннотаций, стандартные аннотации
    a. @Override
    b. @ SuppressWarnings
8. Мета-аннотации
    a. @Retention (RetentionPolicy.SOURCE, RetentionPolicy.CLASS, RetentionPolicy.RUNTIME)
    b. @Target (ElementType.TYPE, ElementType.FIELD и т.д.)

Материалы

Ресурсы
Доступ к ресурсам через Class.getResourceAsStream()
java.util.Properties класс для работы с текстовым файлом формата ключ = значение.

```
static{
        try{
            InputStream in = MyClass.class.getResourceAsStream(«some.properties»);
            p.load(in);//load(Reader), loadFromXML(InputStream)
            in.close();
        }catch(IOException e){}
}
```

Proxy

```
Foo f = (Foo) Proxy.newProxyInstance(Foo.class.getClassLoader(),
                                     new Class[] { Foo.class },
                                     handler);
```

Где InvocationHandler: *Object invoke(Object proxy, Method method, Object[] args)*

AccessibleObject.setAccessible

Доступ до защищенных полей

```
A a = new A();
Field x = a.getClass().getDeclaredField("x");
x.setAccessible(true);
```

Аннотации

```
//для доступа во время исполнения
@Retention(RetentionPolicy.RUNTIME)
@Target (ElementType.FIELD)
@interface Anno {
int length();
}
```

Объявление

```
class SomeClass {
@Anno( length = 100)
private String name;
```

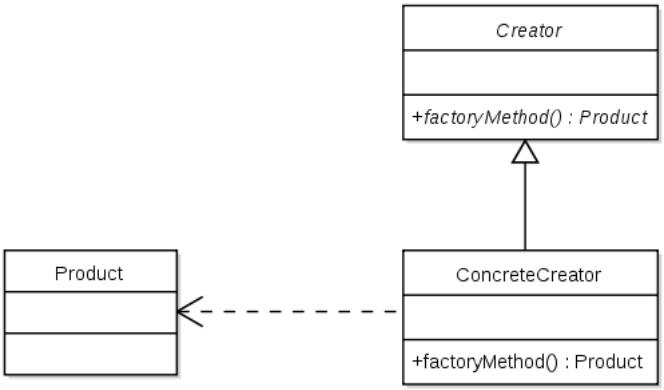
Доступ

```
Class c = ob.getClass();
Field f = c.getField("name");
Anno anno = f.getAnnotation(Anno.class);
```

Factory method pattern

http://en.wikipedia.org/wiki/Factory_method_pattern#Java

It deals with the problem of creating objects (products) without specifying the exact class of object that will be created. The factory method design pattern handles this problem by defining a separate method for creating the objects, which subclasses can then override to specify the derived type of product that will be created.



Dependency injection

http://ru.wikipedia.org/wiki/Внедрение_зависимости

Условно, если объекту нужно получить доступ к определенному сервису, объект берет на себя ответственность за доступ к этому сервису: он или получает прямую ссылку на местонахождение сервиса, или обращается к известному «сервис-локатору» и запрашивает ссылку на реализацию определенного типа сервиса. Используя же внедрение зависимости, объект просто предоставляет свойство, которое в состоянии хранить ссылку на нужный тип сервиса; и когда объект создается, ссылка на реализацию нужного типа сервиса автоматически вставляется в это свойство (поле), используя средства среды.

Java — задания

Развитие калькулятора с семинара №2.

Создать класс фактори для создания команд. То есть создать отельный класс который будет иметь с аргументом имя команды и возвращать реализацию интерфейса Command для данной команды. Реализовать следующие возможности:

1. Конфигурация списка команд через property файл factory класса. Это позволит добавлять команды без перекомпиляции остального проекта. Добавить новые арифметические команды: EXP, LOG
2. Реализовать *Dependency injection* для команд калькулятора: добавить аннотацию @In(arg={STACK, CONTEXT}) для полей классов реализующих команды калькулятора. При создание команды в фактори классе проверять наличие подобных аннотаций и устанавливать данные поля до первого вызова команды. Реализовать STACK, CONTEXT как перечисление (Enum)
3. **Дополнительно**
   Добавить режим «отладки» для команд: в данном режиме фактори возвращает не саму команду, а Proxy объект (см java.lang.reflect.Proxy) в котором в файл журнала средствами log4j пишется:
   1. Stack before
   2. Context
   3. Arguments
   4. Stack after