

Раскроем карты

Андрей Терешко

Пора, наверное, пора. Уж скоро год, как я задал свои загадки (см. "[Ловкость рук и никакого...](#)" и "[Волшебство](#)"). Пришло и несколько писем с ответами, что удивительно. :-)

Вопрос: [На этот вопрос, идею которого подсказал Роман Поротников, правильно ответил Andrei Prygounkov.]

Измените класс Quirk так, чтобы, не изменяя (а также, не удаляя и не добавляя) ни одного символа в коде метода main, на консоль выводилось бы другое сообщение, вместо "Magic Java.":

```
public class Quirk {
    public static void main(String[] args)
    {
        System.out.println("Magic
Java.");
    }
}
```

Ответ: Действительно, так как при загрузке класса первым начинает выполняться не метод "static void main(String[] args)", как многие возможно считают, а блоки статической инициализации, то именно в них и можно вывести на консоль другое сообщение и вообще - завершить программу так, что метод "static void main(String[] args)" вообще не будет никогда вызван!

// Обратите внимание, в КОДЕ метода " main(String[] args)" я не изменил ни одного символа. :-)

```
public class Quirk {
    public static void main(String[] args)
    {
        System.out.println("Magic
Java.");
    }
}
```

```

    }
    static {
        System.out.println("Cool
Java.");
        System.exit(0);
    }
}

```

Вопрос: ["Зеленая" Java. Эта загадка от Романа Поротникова не нашла решения ни у одного из приславших мне письма с ответами. Неужели эта зелень НИКОМУ не по зубам? ;-)]

В JBuilder (да и не только в нем) есть синтаксический анализатор кода, который показывает зеленым цветом код комментария. Напишите программу, которая вся(!) была бы окрашена в зеленый цвет: Green.java. И эта "зеленая" программа должна выводить на консоль сообщение "World is green!" — вот "зеленые" то обрадуются.

Ответ: Конечно, вопрос довольно эффектный и многих может ввести в шок. Как так? — грузим исходник в редактор, видим одну зелень, то есть ВСЯ программа визуально(!) состоит из ОДНИХ комментариев, запускаем программу и, на тебе, — программа работает и выводит на консоль сообщение "World is green!" Чудо да и только! А секрет прост, ведь текст в Java можно набирать не только на английском, китайском, тайваньском и русском, но и на всех этих (и многих других) языках разом, используя ... Эка, я Вам уже сколько подсказал. Но задача то красивая — вся такая зеленая. Нет, пока секрет полностью не открою. Присылайте решения. ;-)

Вопрос: Конструктору класса нельзя приписать атрибут "synchronized", ибо, как пишут **К.Арнольд и Д.Гослинг ("Язык программирования Java")** - *"Конструктор не обязан быть synchronized, поскольку он выполняется только при создании объекта, а это может происходить только в одном потоке для каждого вновь создаваемого объекта"*. Замечу, что не просто "не обязан", а просто и не может иметь атрибут synchronized, иначе при компиляции сразу вылетает ошибка: "Constructors can't be native, abstract, static, synchronized, or final". Приведите пример, возможно и гипотетического, но вполне работоспособного кода, когда конструктор просто обязан(!) обладать свойствами, обычно даруемыми атрибутом "synchronized". И как же этого достичь?

Ответ: Если конструктор в теле своего кода вызывает метод какого либо класса,

Раскроем карты

передавая ему в качестве параметра САМОГО СЕБЯ (this) и команды тела конструктора продолжают выполняться дальше, то существует вероятность в многопоточном приложении, что ДРУГОЙ поток сумеет обратиться к синхронизируемому методу объекта, ВСЕ ЕЩЕ создаваемого этим конструктором, ДО ТОГО КАК конструктор успеет выполниться до конца. Тем самым произойдут незапланируемые программистом действия. В этом случае, то есть в том случае, если в конструкторе происходит "выброс во вне" ссылки на еще не полностью проинициализированный объект (то есть "this"), надо использовать внутри конструктора ОПЕРАТОР synchronized (this):

```
public class Client {
    public Client() {
        synchronized (this){
            Manager.add(this); // "Выброс себя
вовне"
            ... // Различные действия по инициализации полей
объекта (возможно и длительные)
        }
    }
    synchronized void addAccount(){
        ...
    }
} // end Client
```

И пока в конструкторе все еще инициализирующейся объект заблокирован оператором "synchronized (this)" ни один другой тред не сможет вызвать метод "synchronized void addAccount()" этого объекта, хотя ссылка на объект уже передана "в мир" (в данном случае, вызовом static метода класса Manager — Manager.add(this)). А как потом "в миру" будут использовать ссылку на Ваш объект — это, в общем случае, уже может абсолютно не зависеть от Вас.

Вопрос: При вызове конструктора оператором "new", конструктор всегда возвращает ссылку на НОВЫЙ объект. Заставить конструктор вернуть вместо ссылки на новый объект ссылку на уже существующий объект нельзя. А если очень хочется? Как "обмануть" конструктор?

Приведите пример кода, когда при "создании" объекта — то есть при присвоении новой ссылке значения, эта новая ссылка получает значение ссылки на уже существующий объект.

Ответ: Конструктор "обмануть" мне не удалось, как я не старался. ;-)

Но в Java **НОВЫЙ** объект можно создавать(!) не только с помощью оператора "new", но и восстанавливая объект "из пепла" — то есть десериализуя его. И вот в методе readResolve(), при десериализации, можно подставить вместо "себя" другой объект, что и применяется при работе с синглтонами. Если Вы не знаете, что такое "синглтон", то я об этом напишу в следующий раз — ибо это очень интересный объект, почти никак официально не поддерживаемый в спецификации языка Java, но занимающий пристальное и постоянное внимание многих java-программистов. Об этом объекте можно написать целую книгу — и постоянно ее дополнять. ;-).

```
public class Singleton implements Serializable
{
    private static final Singleton INSTANCE = new
Singleton();
    private Singleton( ) { }
    public static Singleton getInstance( )
{
    return INSTANCE;
}
    private Object readResolve() throws
ObjectStreamException {
    return INSTANCE;
}
}
```

Вопрос: В JSP при монтировании в страницу bean-компонента с помощью тега <jsp:useBean ... </jsp:useBean>, используется такой jsp-механизм, что если данного экземпляра bean-компонента, отвечающего требованиям тега, не находится, то с помощью безаргументного конструктора создается новый экземпляр данного bean-компонента.

А если bean-компонент не имеет безаргументного конструктора? Или Вы считаете, что bean обязан всегда иметь безаргументный конструктор?

Укажите, напишите пример, каким образом bean-компонент, не имеющий безаргументного конструктора, можно вмонтировать в jsp-страницу. Сколько имеется способов для этой процедуры?

Ответ: Несмотря на широко распространенное мнение о том, что bean-компонент **ОБЯЗАН** иметь безаргументный конструктор, должен Вас разочаровать — никто этой

Раскроем карты

обязанности ему не вменял. Таким образом, если у Вас есть bean-компонент не имеющий безаргументного конструктора, и Вы желаете его вмонтировать в jsp страницу, то:

Первый способ, используя скриплеты:

```
package bar;
public class FooBean {
    public FooBean(SomeObj arg) {
        ...
    }
    //getters and setters here
}
```

Встраиваем в сессию:

```
<%
someObj x = new SomeObj (...);
bar.FooBean foobar = new FooBean(x);
session.putValue("foobar", foobar);
%>
```

И можем с любой страницы сессии обратиться:

```
<% bar.FooBean foobar =
session.getValue("foobar"); %>
```

Можно встроить и в приложение:

```
<% application.setAttribute("foobar", foobar);
%>
```

Или только в область запроса:

```
<% request.setAttribute("foobar", foobar);
%>
```

Если не размещается в request, session или application scope, то можно использовать только в пределах jsp-страницы: page scope

Как только bean встроен, к нему можно обращаться как обычно:

```
<jsp:getProperty name="foobar"
property="someProperty"/>
<jsp:setProperty name="foobar"
property="someProperty" value="someValue"/>
```

Второй способ - сериализуйте этот bean-компонент и уже в таком, сериализованном виде вмонтируйте в jsp страницу:

```
<jsp:useBean id="shop"
type="shopping.Disks" beanName="Disks" />
<jsp:getProperty name="shop"
property="album" />
```

В beanName вместо "filename.ser", указать "filename".

Вопрос: Модификатор поля (переменной класса) "final" — "объявляет, что значение переменной присваивается всего один раз, при ее инициализации" (**К.Арнольд и Д.Гослинг "Язык программирования Java"**). Приведите пример, когда волшебным образом, можно менять поле с модификатором "final" столько раз, сколько душе угодно. Подсказка — это поле вы, уважаемый Читатель, наверняка знаете и довольно часто используете в своих программах на Java.

Ответ: Пришло несколько писем, где правильно указали имя этого широко известного и повсеместно всеми применяемого и изменяемого(!) поля с модификатором "final". Но никто не указал — почему это происходит? Почему вообще возможно "переписывать финал" в java. Кто и что позволяет это сделать? А как Вы считаете, лично Ваши переменные с модификатором поля "final" кто-либо может изменить? И если да, то каким именно способом?

Вопрос: Если, при создании класса, приписать методу атрибут доступа "private", то доступ к такому методу может быть осуществлен только из самого класса. Приведите пример, когда разработчик класса должен создавать методы класса с атрибутом доступа "private", но нигде(!) в своем классе, не использовать вызов этих методов. Вот как удивлялся этому волшебству **Д. Флэнэген** в своей книге **"Java in a Nutshell"**: "... Как ни странно, но эти методы не определены ни в одном из интерфейсов. Их следует объявлять как "private", что также вызывает некоторое удивление, поскольку они вызываются извне...". Кому же нужны такие методы? Кто их вызывает извне и как?

Ответ: На это вопрос также поступило несколько писем, где правильно — когда именно разработчик класса должен создавать методы класса с атрибутом доступа "private", но нигде(!) в своем классе, не использовать вызов этих методов. Но никто не написал — почему вообще такое возможно? И можете ли Вы вызывать методы другого класса, определенные с атрибутом доступа "private"? И если да, то каким именно

Раскроем карты

способом?

Полный ответ на этот вопрос такой же и как на предыдущий. Все очень просто. ;-) Отгадка механизма этого "фокуса" похожа на разоблачение фокусника, достающего кролика из совершенно пустого ящика. И когда зрителю раскрывают секрет этого фокуса, то по зрительному залу проносится вздох: "Я так и знал, что у этого ящика есть двойное дно!". ;-)

Затем интерес к фокусу пропадает и фокуснику приходится придумывать новый фокус, используя тот же механизм, но в другой упаковке — он начинает, например, распиливать женщину. ;-)

Так и в Java — почти все "фокусы" и "некоторые удивления" опираются на то, что ... Ну, я немного устал, да и Вы, уважаемый читатель, немного устали читать. Все, что не успел, я сообщу Вам в следующий раз. Что-то ведь должно остаться еще некоторое время неразгаданным для широкой публики. ;-)

Успехов Вам.

Все права на публикацию этого материала принадлежат автору.

Терешко Андрей, Минск.