Лабораторная 2. Основы объектно-ориентированного проектирования в Java.

Цель:

- создание приложений, реализующих различные операции над элементами массивов.
- получение навыков объектно-ориентированного анализа и проектирование классов для задач из различных предметных областей;

Упражнение 1. Использование аргументов командной строки и разбор массива args.

В этом упражнении реализуется приложение, которое принимает параметр из командной строки и выводит сообщение, относящееся к нему. Вы создаете класс, который принимает аргумент, значение которого лежит в интервале от 1 до 5 включительно. В программе случайным образом генерируются числа от 1 до 5. Если число, сгенерированное в программе, равно числу, введенному из командной строки, то на экран выводится сообщение о выигрыше.

Для решения поставленной задачи:

1. Создайте новый Java класс и назовите его GuessingGame.
2. В методе main, объявите две переменные типа int: randomNum и guess.
3. Добавьте в код метода main сообщение с информацией о допустимых значениях аргументов, если при вызове приложения не было введено ни одного аргумента или было введено слово “help”.

   Основные шаги по реализации программы описаны в псевдокоде:

```
if length of args array == 0 or value of args[0] = “help”
print a Correct Usage message

else
randomNum = a generated random number 1 - 5
guess = integer value of args[0]

if argument < 1 or > 5
print an error message (invalid argument)

else
if argument == randomNum
print congratulations message

else
print a “Sorry; try again” message
```

4. Для того чтобы сгенерировать произвольное число от 1 до 5, воспользуйтесь следующим выражением:

   ```
   (int)(Math.random()*5+1);
   ```

5. Для преобразования элемента массива типа String в тип int используйте вызов метода parseInt класса Integer. Проверьте работу приложения.

Упражнение 2. Создание пользовательского класса и включение комментариев в код

1. Выполните объектно-ориентированный анализ банковской системы и спроектируйте требуемые классы.

Требуется: провести анализ предметной области и спроектировать классы.

Решение:

Пользовательскими классами, необходимыми для создания банковского приложения являются:

- Account. Класс Account является абстракцией счета в банке. Всякий раз, когда банковское приложение нуждается в новом счете, объект Account (конкретный счет) создается, используя конструктор класса Account.
- ATM. Класс ATM является абстракцией банкомата (ATM). Банковское приложение создает одиночный экземпляр класса ATM. Например, ATM обеспечивает интерфейс, требуемый клиентам банка для получения информации о своих текущих операциях.
- SimpleBankApp. Класс SimpleBankApp является классом, содержащим точку входа в банковское приложение.

2. Спроектируйте класс Account.

Для выявления особенностей класс Account необходимо ответить на следующие вопросы:

- Что представляет класс?

Любой класс Java представляет собой новый сложный пользовательский тип данных. Класс Account (Счет) является типом, который представляет и определяет банковский счет.

- Что представляет экземпляр класса?

Экземпляр класса представляет собой виртуальную сущность любого класса (типа), т.е. выделенную и определенную память, размер которой зависит от типа данных, определяемого описанием класса. Экземпляр класса Account представляет собой виртуальную сущность банковского счета.

- Какая информация содержится в классе?

Класс содержит два типа информации:

- ✓ атрибуты типа.

Каждый пользовательский тип имеет дополнительный набор атрибутов. Атрибутами класса Account (счет) являются `customer` (клиент) и `balance` (баланс). Атрибуты объявляются как поля в классе с использованием пары: имени и типа данных.

Определите тип, модификатор доступа и назначение атрибутов для класса Account.

- ✓ операции типа

Операции типа выражаются с помощью методов, объявляемых в классе. Ниже перечислены методы класса Account:

Метод	**Описание**

getBalance	Возвращает текущий баланс.
deposit	Добавляет сумму к балансу (Учтите возможность ввода копеек).
withdraw	Вычитает сумму из баланса.
getCustomer	Возвращает ID клиента .
getDetails	Возвращает дополнительную информацию о счете. Может содержать имя клиента, текущий баланс и т.д.

3. Реализуйте класс Account, методы и атрибуты которого перечислены в п.2. Если необходимо, то дополните класс собственными атрибутами и методами. Не забывайте о принципе инкапсуляции.

4. Реализуйте основной класс, который является точкой входа для приложения. Основной класс должен иметь основной метод `main`, в котором объявляется и инициализируется объект типа Account. В методе main необходимо продемонстрировать все возможности работы с объектом типа Account.

5. Постройте и протестируйте приложение.

6. Создайте класс ATM, в котором реализуйте метод Out для вывода информации о текущем состоянии счета, текущий счет передается в метод в качестве параметра. В тело метода перенесите требуемые строки из метода main, а в методе main обеспечьте вывод информации о счете, используя метод Out.

7. Постройте и протестируйте приложение.

8. Реализуйте следующую возможность: при создании объекта класса Account на счет кладется некая начальная сумма (присваивается значение полю balance).

9. Постройте и протестируйте приложение.

10. Используя опцию -d для утилиты javac, откомпилируйте весь проект и сохраните его в папке, например c:\1\

11. Запустите откомпилированный проект, используя командную строку.

Разрабатывая программу, можно документировать информацию о пакетах, классах, методах и полях. Комментарии помещаются непосредственно перед элементом, к которому они относятся. Следует учитывать, что по умолчанию документация создается только для элементов, имеющих уровень видимости `public` или `protected`.

Комментарии, имеющие вид /** ... */, содержат произвольный текст, содержащий *дескриптор* (tag). Дескриптор начинается символом, например `@author` или `@param`. Первое предложение текста комментариев должно представлять собой краткое описание.

Утилита **javadoc** автоматически генерирует страницы, состоящие из кратких описаний.

В самом тексте можно использовать элементы языка HTML, например,

. . . — для выделения текста курсивом,

<code>. . .</code> — для установки моноширинного шрифта,

. . . — для выделения текста полужирным шрифтом,

<img . . . > — для вставки рисунков.

12. Добавьте комментарии к классу Account. Комментарии к классу должны помещаться после директив import, непосредственно перед определением класса. Например:

```
/**
Объект класса <code> Account </code> имитирует банковский счет
*/
public class Account {
}
```

13. В разделе “author” добавьте информацию об авторе, например, свое имя, скорректировав дескриптор @author.

14. После данных об авторе добавьте информацию о версии, используя дескриптор @version «текст». В данном случае «текст» означает произвольное описание текущей версии.

15. Добавьте комментарии к методам. Комментарий должен предшествовать конкретному методу, который он описывает. Например, добавьте к методу withdraw() следующий комментарий:

```
/**
Метод <em> withdraw <em> обеспечивает снятие денег со счета
*/
```

16. Кроме дескрипторов общего назначения можно использовать специальные дескрипторы:

`@param переменная описание`

Этот дескриптор добавляет в описание метода раздел “parameters” (“параметры”), т.е. информацию о параметре метода.

`@return` *описание*

Данный дескриптор добавляет в описание метода раздел “returns” (“возвращаемое значение”), т.е. информацию о возвращаемом методом значении и его типе.

`@throws` `описание_класса.`

Этот дескриптор описывает информацию об исключительных ситуациях, которые могут генерироваться методом.

17. Добавьте к методу deposit()комментарий с дескриптором @param.

18. К методу getBalance() добавьте комментарий с дескриптором @ return.

19. В командной строке вызовите утилиту javadoc –d и изучите ее параметры.

20. Перейдите в каталог, содержащий исходный файл Account.java, подлежащий документированию.

21. Для документирования исходного файла выполните команду javadoc (в отчете необходимо привести эту команду с параметрами).

22. В проводнике откройте папку и просмотрите сгенерированные HTML-документы.

При вызове `javadoc` можно указывать различные опции. Например, чтобы включить в документацию дескрипторы **@author** и **@version**, можно использовать опции `-author` и `-version`.

23. Проще сгенерировать файл документации в среде разработки NetBeans. Выберите пункт меню Build → Generate Javadoc (Выполнить → Создать документацию Java). В этом случае запускается создание документации по проекту. При этом из исходных кодов классов проекта выбирается информация, заключенная в документационные комментарии /** … */, на ее основе создается гипертекстовый HTML-документ. Посмотрите документ и найдите в нем свои комментарии.

24. Для настройки процесса документирования откройте окно свойств проекта, и выберите раздел Документирование. В этом разделе в пункте «Документировать дополнительные теги» включите теги @author и @version.

Упражнение 3. Отладка приложений в интегрированной среде разработки.

Практически все интегрированные среды разработки в свой состав включают отладчик - Debugger. Как правило, в независимости от среды разработки, отладчики реализуют схожий функционал.

В этом упражнении Вы научитесь пользоваться отладчиком интегрированной среды разработки NetBeans.

1. Спроектируйте класс Shirt (рубашка) в интегрированной среде разработки NetBeans.

   Атрибутами класса Shirt (рубашка) являются:

   - shirtID – идентификатор рубашки, проинициализированный значением 0;
   - description – описание рубашки, проинициализированное значением "description";
   - colorCode – цвет рубашки, проинициализированный значением из списка: ‘R’=Red - красный, ‘B’=Blue - синий, ‘G’=Green - зеленый, ‘U’=Unset – не установленный;
   - price – цена рубашки, проинициализированная значением 0.0;
   - quantityInStock – количество на складе, проинициализированное значением 0.

   Определите тип атрибутов.

Методом класса является метод displayShirtInformation(), который позволяет выводить на экран информацию о конкретном экземпляре класса Shirt.

Вывод информации на экран реализуется с использованием метода println объекта out класса System.

Метод имеет следующий вид:

```
public void displayShirtInformation() {
```

```
        System.out.println("Shirt ID: " + shirtID);
        И .т.д. по всем параметрам.
    }
```

Дополните метод, чтобы вся информация об объекте выводилась на экран.

2. Реализуйте основной класс (ShirtTest), который является точкой входа в приложение. Основной класс должен иметь метод main:

```
public static void main(String[] args) { // code goes here }
```

Главный класс приложения имеет следующий вид:

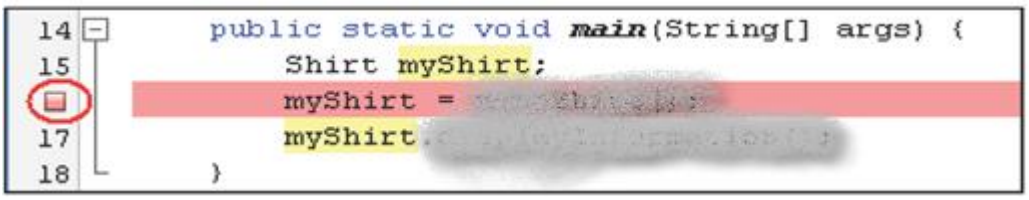
```
public class ShirtTest
{
        public static void main (String args[]) {
         3 инструкции;
        }
}
```

В методе main создайте 3 инструкции:

1. Объявите переменную myShirt, которая будет хранить ссылку на конкретный экземпляр класса Shirt.
2. Проинициализируйте переменную myShirt, вызвав конструктор соответствующего класса.
3. Вызовите метод, позволяющий получить всю информацию о конкретной рубашке.

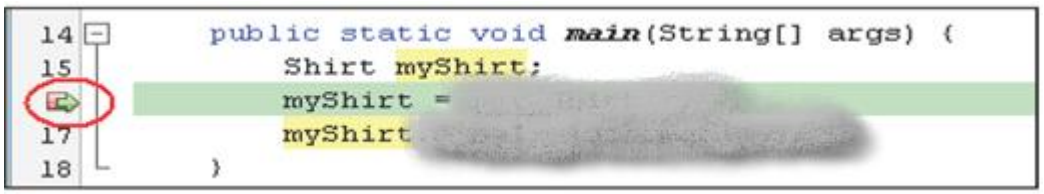
3. Установите точку останова (breakpoint) в классе ShirtTest, нажмите левой кнопкой мыши в левом поле редактора кода напротив той строки, с которой вы хотите производить отладку. В нашем упражнении этой стройкой будет строка, в которой создается объект рубашка, т.е. строка в которой вызывается конструктор по умолчанию.

Красный квадрат появится в левой части редактора кода, который символизирует точку останова.



Создайте точку останова для строки, в которой производится вызов метода displayShirtInformation;

4. Запустите отладку, нажав правой кнопкой мыши на файл ShirtTest в окне проектов, выберете пункт отладка. Или в меню Отладка->Отладка файла.
5. Отладчик начнет выполнение программы и остановится в месте точки останова. В панели редактора кода измениться пиктограмма красного квадрата на квадрат со стрелкой, и строка с точкой останова подсветиться в зеленый цвет.



Обратите внимание, что данная строка кода еще не выполнена.

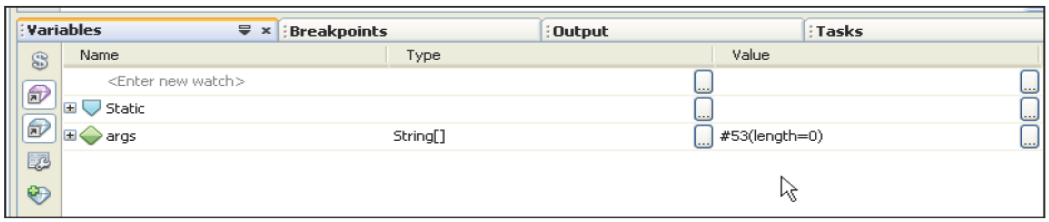
6. При запуске отладчика с окнами NetBeans произойду некоторые изменения.

- Появиться новая панель инструментов, содержащая кнопки, которые позволяют управлять режимом отладки.

Наведите курсор мыши на каждую из этих кнопок и прочитайте, для чего данные кнопки предназначены.

- ✓ Первая кнопка «Завершение сеанса отладчика» – останавливает отладку.
- ✓ Вторая «Приостановить» – прерывает выполнение отладчика.
- ✓ Третья кнопка «Продолжить» - обеспечивает переход к следующей точке останова или выполняет до конца программу, если таких точек нет.
- ✓ Четвертая «Перешагнуть» – переходит к выполнению следующей строчки в программе в текущем классе. В нашем случае производиться переход к строчке вызова метода displayShirtInformation.
- ✓ Пятая «Перешагнуть выражение» – позволяет переходить к выполнению следующего выражения без захода в текущее. Например, позволяет не заходить в циклы при отладке.
- ✓ Шестая кнопка «Войти» – позволяет входить в другие блоки (методы классов), ссылки на которые присутствуют в текущем коде.
- ✓ Седьмая «Выйти» – позволяет возвращаться в блок (метод класса) из которого был запущен отладчик.
- ✓ Восьмая «Выполнить до курсора» – отладка выполняется до той строчки кода, в которой установлен курсор.

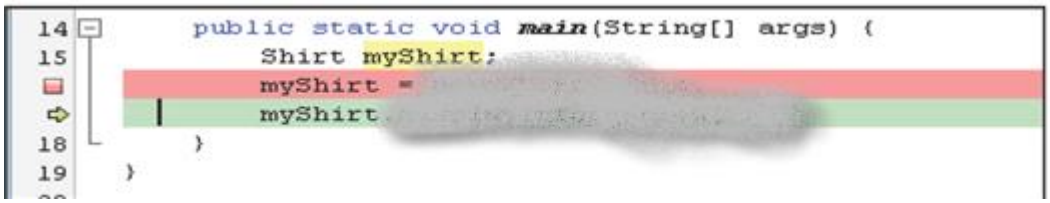
- Следующим изменением, связанным с запуском отладчика, будет изменение нижней части окна. В ней появится панель, позволяющая контролировать изменения значения переменных и точки останова.



Во вкладке «Переменные» появятся переменные, доступные в данном классе. Помните, что выполнение на данном шаге приложения остановилось до создания объекта типа Shirt, поэтому вы не можете видеть атрибуты класса Shirt.

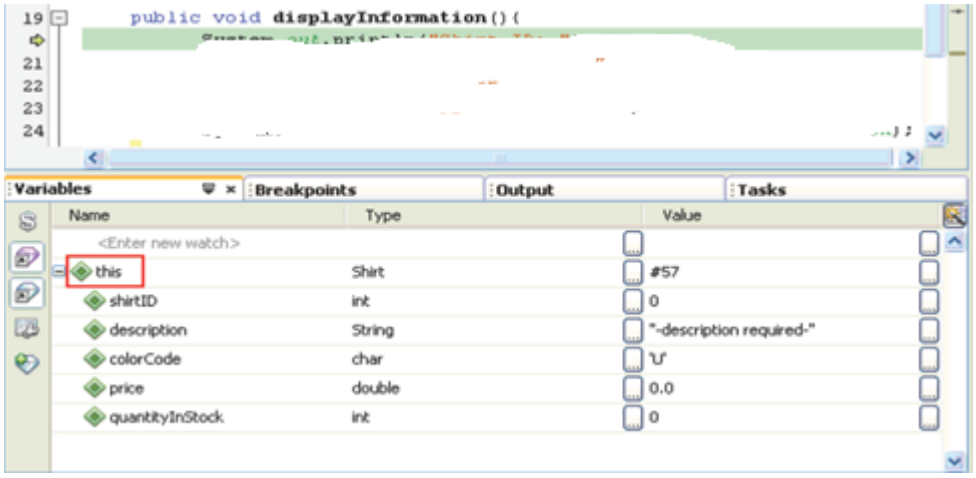
7. Нажмите на кнопку «Перешагнуть» и перейдите к следующей строчке кода.

8. Стрелка указывает на строку, в которой вызывается метод displayShirtInformation объекта myShirt. Во вкладке «Переменные» вы можете увидеть атрибуты объекта myShirt.

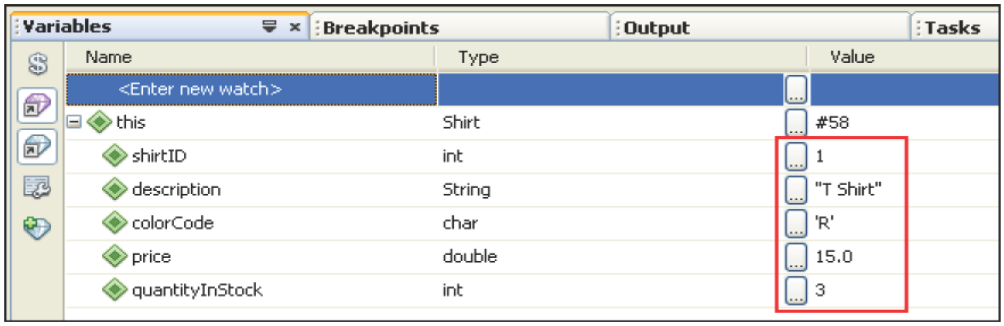


На данном шаге метод displayShirtInformation() еще не был выполнен. Вы можете изменить значения полей объекта myShirt, используя окно «Переменные».

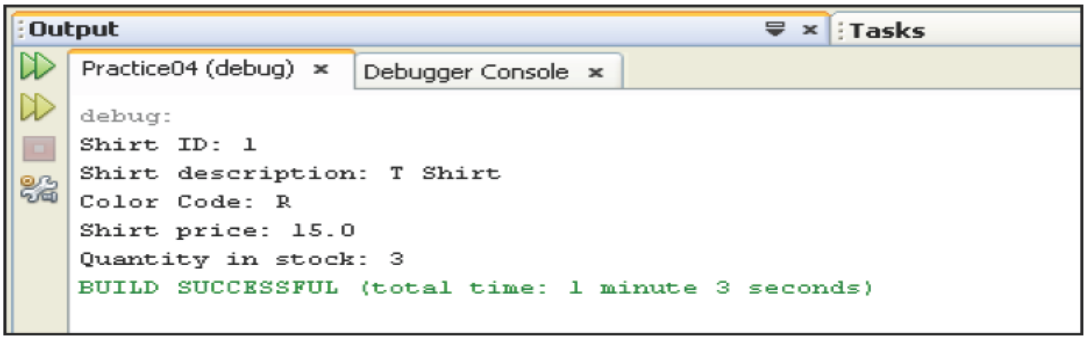
Для того, чтобы в режиме отладки зайти в метод displayShirtInformation(), нажмите кнопку «Войти». В редакторе кода откроется окно, содержащее класс Shirt и стрелка отладчика будет указывать на первую строку метода displayShirtInformation().



В окне «Переменные» в дереве раскройте ветку this и измените значения атрибутов объекта myShirt. Для сохранения изменения значения используйте клавишу «Tab».



9. Нажмите на кнопку «Выйти» и выполните до конца приложение.
10. Откройте окно вывода результатов:



Обратите внимание, что значение атрибутов объекта myShirt изменились на новые, которые были введены в окне «Переменные».