Лабораторная работа 3
Основы работы с классами и объектами

Цель

Ознакомиться на практике с основой работы с классами и объектами: объявление объектов, представление методов, конструкторы, перегрузка методов, управление доступом, вложенные и внутренние классы.

Задание на лабораторную работу

1. Создайте класс «Шар». Класс должен иметь свойство «Размер» и конструктор для установки размера при создании. Изменять размер шара после создания нельзя.

2. Создайте класс «Контейнер». В контейнер должны помещаться шары. В контейнере не должно быть одинаковых шаров (т.е. необходимо осуществлять проверку при добавлении шара на наличие его в контейнере).

Контейнер должен включать следующие методы:

- Добавление шара
- Удаление шара
- Подсчет количества шаров в контейнере
- Подсчет суммарного размера шаров в контейнере
- Очистка контейнера
- Поиск шара в контейнере (проверка наличия)
- Вывод размеров всех шаров в контейнере

3. Создайте класс «Ящик», являющийся наследником класса «Контейнер» и отличающийся от него ограниченным размером. Размер ящика должен задаваться при его создании (в конструкторе). Изменять размер ящика после создания нельзя. Если при добавлении шара будет происходить превышение размера ящика, добавления происходить не должно. Класс должен содержать метод, возвращающий список шаров, отсортированный по размеру.

4. Создайте приложение, демонстрирующее работу класса «Ящик»

5. Создайте набор классов для лабиринта

- Лабиринт состоит из комнат, соединенных проходами.
- Лабиринт может включать в себя входы, выходы,
- В комнатах могут храниться предметы,
- По лабиринту могут передвигаться люди, которые эти предметы могут забирать.
- В лабиринт можно войти, можно ходить из комнаты в комнату, забирать предметы, видеть то, что находится рядом, из лабиринта можно выйти.

6. Создайте приложение, демонстрирующее работу набора классов «Лабиринт».

Теоретический материал

Основные базовые блоки языка Java - классы. Схема синтаксиса описания класса такова:

[Модификаторы] class ИмяКласса [extends ИмяСуперкласса] [implements ИменаИнтерфейсов]

{

  Поля класса;

  Методы;

}

где:

Модификаторы - ключевые слова типа ***static, public*** и т.п., модифицирующие поведение класса по умолчанию;

ИмяКласса - имя, которое вы присваиваете классу;

ИмяСуперкласса - имя класса, от которого наследуется ваш класс;

ИменаИнтерфейсов - имена интерфейсов, которые реализуются данным классом.

Схема описания методов:

[Модификаторы] ВозвращаемыйТип ИмяМетода ([Список Параметров])

{

  [Тело Метода]

}

Для начальной инициализации класса используются специальные методы, называемые конструкторами. Имя конструктора совпадает с именем класса; конструкторы не имеют никакого возвращаемого значения (тип возвращаемого значения не указывается, даже в случае ***void***). Класс может иметь несколько конструкторов, отличающихся набором параметров (например, конструктор без параметров для инициализации полей класса значениями по умолчанию и конструктор с параметрами - значениями для соответствующих полей класса).

Для инициализации объекта класса необходимо использовать оператор ***new*** в простейшей форме:

ИмяКласса ИмяПеременной = new ИмяКласса();

либо (раздельное объявление объектной переменной и ее инициализация)

ИмяКласса ИмяПеременной;

ИмяПеременной = new ИмяКласса();

где

ИмяПеременной – имя объектной переменной.

Если в операции new не указано параметров, то для инициализации класса будет использован конструктор по умолчанию, в противном случае будет вызван конструктор с соответствующим набором параметров.

Пример класса и его использования:

```
public class Example {

	// Поле класса
	private int integerField;

	//Конструктор по умолчанию
	public Example() {
		integerField = 0;
	}

	//Конструктор с параметром
	public Example(int integerValue) {
		setIntegerField(integerValue);
	}

	//Метод установки значения поля
	public void setIntegerField(int newValue) {
		integerField = newValue;
	}

	//Метод, возвращающий значение поля
	public int getIntegerField() {
		return integerField;
	}
}

public class ExampleDemo {
	public static void main(String[] args) {
		//Создание экземпляра класса Example
		Example exampleInstance = new Example();

		//Создание еще одного экземпляра класса Example
		Example anotherExampleInstance = new Example(1000);

		//Вывод на экран значения поля объекта exampleInstance
		System.out.println(exampleInstance.getIntegerValue());

		//Установка нового значения поля объекта exampleInstance
		exampleInstance.setIntegerValue(50);
```

```
            //Вывод на экран значения поля объекта exampleInstance
            System.out.println(exampleInstance.getIntegerValue());

            //Вывод на экран значения поля объекта exampleInstance
            System.out.println(anotherExampleInstance.getIntegerValue());
        }
    }
```

Примечания

Не забывайте, что каждый класс должен быть создан в отдельном файле.

Для хранения списка шаров в контейнере используйте реализацию стандартного класса-коллекции (например, ***java.util.LinkedList*** или ***java.util.ArrayList***). Подробную информацию о классах-коллекциях и их использовании смотрите в документации, либо в Java Tutorial, раздел ***Collections***.

Также см. прилагаемый пример ***WordsList.java.***

Для последовательного обхода всех элементов коллекции в Java может использоваться особая форма цикла ***for*** (называемая также ***for-each***):

```
for (ТипЭлемента элемент: Коллекция)
{
  doSomething with элемент…
}
```

Пример использования см. в WordsList.java

При наследовании (директива ***extends*** при определении класса) класс-потомок наследует все поля и методы класса-родителя (за исключением объявленных как ***private***), поэтому не требуется описывать те же действия в унаследованном классе. Если какой-либо метод в классе-потомке должен работать иначе, чем в родительском классе, его можно переопределить (создав метод с тем же именем и параметрами в классе-потомке); при использовании объекта класса-потомка будет вызван именно переопределенный метод.