JAVA. Лабораторная работа №3
Тема: Введение в ООП

1. Цель работы

Получить основные понятия по следующим разделам языка Java:

– объектно-ориентированное программирование;

– создание собственных классов.

2. Теоретический материал

Объектно-ориентированное программирование (ООП) в настоящее время стало доминирующей парадигмой программирования, вытеснив «структурные», процедурно-ориентированные подходы, разработанные в 1970 годах.

Java представляет собой полностью объектно-ориентированный язык.

Объектно-ориентированная программа состоит из объектов. Каждый объект имеет определенную функциональность, которую предоставляет в распоряжение пользователей, а также скрытую реализацию. Многие объекты программ могут быть взяты программистами в готовом виде из стандартных пакетов Java, а некоторые написаны самостоятельно.

Классы и объекты

Класс – это шаблон, или проект, по которому будет сделан объект. При разработке собственных классов необходимо пользоваться абстрагированием от совокупности свойств реальных предметов, и выбирать только те характеристики и свойства предмета, которые удовлетворяют семантике решаемой задачи. Например, если разрабатывается информационная система, которая отвечает за распечатывание квитков при получении сотрудниками заработной платы, то для описания сотрудника в такой системе может быть достаточно, указать ФИО сотрудника, размер и дату выдачи заработной платы. Если же речь идет об информационной системе по регистрации и обслуживанию больных в поликлинике, то значения размера и даты выдачи заработной платы больных не имеют никакого значения, а вот информация о месте их проживания, дате рождения и истории болезни играют существенную роль.

Объявление класса на языке Java может быть сделано следующим образом:

```
модификатор class class_name [extends parent_class] {//тело класса
  //объявление полей класса
  тип  имя_поля; //свойства (поля) класса class_name
  //объявление конструктора класса
  модификатор class_name(аргументы){тело конструктора};
  //объявление методов класса
  модификатор тип method_name (аргументы){тело метода};
}
```

Например, класс сотрудник Employee можно описать следующим образом:

```
public class Employee {
  //перечисление полей класса
  private String name; // имя
  private double salary; // размер заработной платы
  private Date hiredate; // дата приема на работу

  //конструктор класса, задача которого – присвоение значений полям класса
  public Employee(String n, double s, int year, int month, int day){
    name=n;
    salary=s;
    hiredate=(new GregorianCalendar(year,month-1,day)).getTime();
  }

  //методы класса
  public String getName { //возвращает имя сотрудника
    return name
  }

  public double getSalary { //возвращает размер заработной платы сотрудника
    return salary}

  public Date getDate{ // возвращает дату приема на работу
    return hiredate}
}
```

Объект - это мыслимая или реальная сущность, обладающая характерным поведением, отличительными характеристиками и являющаяся важной в предметной области. Объектом является экземпляр класса, созданный путем вызова конструктора класса. Каждый объект обладает состоянием, поведением и уникальностью.

Состояние (state) - совокупный результат поведения объекта, одно из стабильных условий, в которых объект может существовать. В любой конкретный момент времени состояние объекта включает в себя перечень свойств объекта и текущие значения этих свойств.

Поведение (behavior) - действия и реакции объекта, выраженные в терминах передачи сообщений и изменения состояния; видимая извне и воспроизводимая активность объекта.

Уникальность (identity) - природа объекта; то, что отличает его от других объектов.

Объекты и объектные переменные

Чтобы работать с объектами, их нужно сначала создать и задать исходное состояние. Затем к этим объектам применяются методы. В языке Java для создания новых экземпляров используются конструкторы. ***Конструктор*** – специальный метод, предназначенный для создания и инициализации экземпляра класса. Имя конструктора всегда совпадает с именем класса. Следовательно, конструктор класса Employee называется Employee и объявляется как

```
public Employee(String n, double s, int year, int month, int day){ name=n;
  salary=s;
  hiredate=(new GregorianCalendar(year,month-1,day)).getTime();
}
```

В одном классе может быть объявлено несколько конструкторов, если их сигнатуры разные. Например, можно создать конструктор в классе Employee, который принимает только имя сотрудника и устанавливает ему заработную плату 1 условная единица. Дата выхода на работу всем таким сотрудникам будет установлена 31 декабря 2009 года.

```
public Employee(String n){ name=n;
  salary=1;
  hiredate=(new GregorianCalendar(2009,12,31)).getTime();
}
```

Для создания объекта необходимо объявить объектную переменную, затем вызвать конструктор класса. Например, объявим две переменные e1 и e2 с типом Employee. Создадим двух сотрудников в информационной системе с помощью вызова различных конструкторов.

```
Employee e1 = new Employee(“James Bond”, 100000, 1950,1,1);
Employee e2 = new Employee(“James NeBond”);
```

В результате вызова конструкторов сотрудник James Bond был принят на работу 1 января 1950 года с заработной платой 100000 у.е. (ссылка на объект сохранена в объектной переменной e1), сотрудник James NeBond был принят на работу 31 декабря 2009 года с заработной платой 1 у.е. (ссылка на объект сохранена в объектной переменной e2), т.к. для его создания использовался второй конструктор, который принимает одно-единственное значение.

Основные понятия ООП – инкапсуляция, наследование и полиморфизм

Наследование (inheritance) - это отношение между классами, при котором класс использует структуру или поведение другого (одиночное наследование) или других (множественное наследование) классов. Наследование вводит иерархию "общее/частное", в которой подкласс наследует от одного или нескольких более общих суперклассов. Подклассы обычно дополняют или переопределяют унаследованную структуру и поведение.

Расширим класс Employee следующим образом. Необходимо описать класс, экземпляры которого представляли бы менеджера предприятия. Менеджер является таким же сотрудником, однако у него есть дополнительное поле – премия. Соответственно, метод, который возвращал в классе сотрудник Employee размер заработной платы, больше не подходит для менеджера.

```
class Manager extends Employee{ //наследование от класса сотрудник
  private double bonus;  // размер премии
  //конструктор класса
  public Manager (String n, double s, int year, int month, int day){
     super(n, s, year, month, day); // т.к. класс унаследован от другого класса, то
     //первой командой в конструкторе класса-потомка необходимо вызвать
     //конструктор родителя. Т.к. в скобках после super указано 5 аргументов, то
     //будет вызван первый конструктор Employee
  }
}
```

Теперь каждый экземпляр класса Manager имеет 4 поля – name, salary, hiredate, bonus. Определяя подкласс, нужно указать лишь отличия между подклассом (потомком) и суперклассом (родителем). Разрабатывая классы, следует помещать методы общего назначения в суперкласс, а более специальные – в подкласс.

В приведенном выше примере не все методы родительского класса Employee подходят для класса Manager. В частности, метод getSalary() должен возвращать сумму базовой зарплаты и премии. Следовательно, нужно реализовать новый метод, ***замещающий*** (overriding) метод класса родителя. Сделать это можно следующим образом:

```
class Manager extends Employee{
  public void getSalary() {новое тело метода….} // перекрытие (замещение)
  //метода класса родителя
....
}
```

Новый (замещенный) метод будет выглядеть так:

```
public void getSalary(){
  double basesalary=super.getSalary();
  return basesalary+bonus;
}
```

Инкапсуляция (encapsulation) - это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса). При использовании объектно-ориентированного подхода не принято использовать прямой доступ к свойствам какого-либо класса из методов других классов. Для доступа к свойствам класса принято использовать специальные методы этого класса для получения и изменения его свойств.

Открытые члены класса составляют внешний интерфейс объекта. Эта та функциональность, которая доступна другим классам. Закрытыми обычно объявляются все свойства класса, а так же вспомогательные методы, которые являются деталями реализации и от которых не должны зависеть другие части системы. Благодаря сокрытию реализации за внешним интерфейсом класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы.

Полиморфизм (polymorphism) - положение теории типов, согласно которому имена (например, переменных) могут обозначать объекты разных (но имеющих общего родителя) классов. Следовательно, любой объект, обозначаемый полиморфным именем, может по-своему реагировать на некий общий набор операций.

Рекомендации по проектированию классов:

- Всегда храните данные в переменных, объявленных как private;
- Всегда инициализируйте данные;
- Не используйте в классе слишком много простых типов;
- Не для всех полей надо создавать методы доступа и модификации;
- Используйте стандартную форму определения класса;
- Разбивайте на части слишком большие классы;
- Выбирайте для классов и методов осмысленные имена.

3. Порядок выполнения работы (все задания выполняются в консоли)

3.1. Создайте класс Rectangle, представляющий прямоугольник, экземпляры которого обладают четырьмя полями целого типа (x1, y1) (левый верхний угол), (x2, y2) (правый нижний угол). Для данного класса создать три конструктора, которые инициализируют поля следующим образом:

– конструктор принимает 4 параметра целого типа и присваивает их значения полям (x1, y1), (x2, y2);

– конструктор принимает 2 параметра целого типа – ширину и высоту прямоугольника, а левый верхний угол прямоугольника помещает в координату (0,0);

– конструктор не принимает никаких параметров – создает вырожденный прямоугольник с координатами углов (0,0) и (0,0).

В классе Rectangle перегрузить метод ***toString()***, выдающий текущее состояние экземпляра прямоугольника (значение полей). Создать метод ***move (int dx, int dy)***, перемещающий прямоугольник по горизонтали на заданное ***dx***, по вертикали на заданное ***dy***. Создать метод ***minSquare*** (подумать какие входные параметры), возвращающий прямоугольник с минимальной площадью по сравнению с другим прямоугольником.

Протестируйте в ней поведение экземпляров класса Rectangle следующим образом: создайте три объекта Rectangle тремя различными созданными конструкторами, выведите состояние всех трех объектов. Воспользуйтесь вызовом функции ***move(…)*** с различными значениями параметров для каждого объекта и выведите новое положение созданных прямоугольников. Протестируйте работу функции ***minSquare(…)*** на одном примере.

3.2. Расширьте класс Rectangle новым классом DrawRect, у которого есть метод отображения цвета границы прямоугольника – ***draw(String outColor)*** и поле ***outColor*** с типом данных String (название цвета границы прямоугольника) – вводится с клавиатуры. Это поле служит для задания цвета границы прямоугольника.

3.3. Расширьте класс DrawRect новым классом ColoredRect, в котором есть поле ***inColor*** с типом String (название цвета прямоугольника) – вводится с клавиатуры. Метод отображения цветов прямоугольника ***draw(String outColor, String inColor)*** перекрывается следующим образом: информация о цветах прямоугольника отображается следующим образом: граница цветом outColor, внутренность – inColor.

Для тестирования функции ***draw(…)*** выведите прямоугольник при помощи первого конструктора с отображением внутреннего цвета и цвета границы.