Практическая работа №10

Теория по стеку

Напомним, что стек — это структура данных, которая содержит данные по принципу «последним пришел — первым обслужен». У стека есть множество применений. Например, компилятор использует стек для обработки вызовов методов. При вызове метода его параметры и локальные переменные помещаются в стек (*операция «push»*). Когда этот метод вызывает другой метод, параметры и локальные переменные уже нового метода также помещаются в стек. Когда новый метод завершает свою работу и возвращается к вызвавшей его стороне, связанное с ним пространство освобождается из стека (*операция «pop»*).

Для моделирования стеков можно определить класс. Предположим, что стек в качестве данных содержит объекты. Для реализации стека объектов можно использовать класс ArrayList.

Теория по клонированию

Чтобы определить пользовательский класс, который реализует интерфейс Cloneable, этот класс должен переопределить метод clone() в классе Object. В следующем коде определяется класс House, который реализует интерфейсы Cloneable и Comparable.

```
public class House implements Cloneable, Comparable<House> {
  private int id;
  private double area;
  private java.util.Date whenBuilt;

  public House(int id, double area) {
    this.id = id;
    this.area = area;
    whenBuilt = new java.util.Date();
  }

  public int getId() {
    return id;
  }

  public double getArea() {
    return area;
  }
```

```
  public java.util.Date getWhenBuilt() {
    return whenBuilt;
  }

  @Override /** Переопределяет protected-метод clone,
    определенный в классе Object, и расширяет его доступность */
  public Object clone() {
    try {
      return super.clone();
    }
    catch (CloneNotSupportedException ex) {
      return null;
    }
  }

  @Override // Реализует метод compareTo, определенный в Comparable
  public int compareTo(House o) {
    if (area > o.area)
      return 1;
    else if (area < o.area)
      return -1;
    else
      return 0;
  }
}
```

Класс House реализует метод clone() (строчки № 26–33), определенный в классе Object. Заголовок метода clone(), определенного в классе Object, следующий:

```
protected native Object clone() throws CloneNotSupportedException;
```

Ключевое слово native указывает, что этот метод написан не на языке Java, а реализован в JVM для собственной платформы. Ключевое слово protected ограничивает доступ к методу в том же пакете или подклассе. По этой причине класс House должен переопределить этот метод и изменить модификатор доступа на public, чтобы этот метод можно было использовать в любом пакете. Поскольку метод clone(), реализованный для собственной платформы в классе Object, выполняет задачу клонирования объектов, метод clone() в классе House просто вызывает super.clone(). Метод clone(), определенный в классе Object, выбрасывает исключение CloneNotSupportedException, если объект не типа Cloneable. Поскольку мы перехватываем это исключение внутри метода, нет необходимости объявлять его в заголовке метода clone().

Класс House реализует метод compareTo, который определен в интерфейсе Comparable. Этот метод сравнивает площадь двух домов.

Теперь можно создать объект типа House и клонировать его следующим образом:

```
House house1 = new House(1, 1750.50);
House house2 = (House)house1.clone();
```

Объекты house1 и house2 являются разными объектами с одинаковым содержимым. Метод clone() класса Object копирует каждое поле исходного объекта в целевой. Если поле примитивного типа, то его значение тоже копируется. Например, значение площади (типа double) копируется из house1 в house2. Если поле объектного типа, то копируется ссылка на это поле. Например, поле whenBuilt типа Date, поэтому ссылка на него копируется в house2. Следовательно, house1.whenBuilt == house2.whenBuilt равно true, несмотря на то, что house1 == house2 равно false. Это называется поверхностной копией (в отличие от глубокой) и означает, что если поле объектного типа, то копируется ссылка на объект, а не его содержимое.

Чтобы сделать глубокую копию объекта типа House, замените метод clone() в строчках № 26–33 следующим кодом:

```
public Object clone() throws CloneNotSupportedException {
  // Сделать поверхностную копию
  House houseClone = (House)super.clone();
  // Сделать глубокую копию whenBuilt
  houseClone.whenBuilt = (java.util.Date)(whenBuilt.clone());
  return houseClone;
}
```

или

```
public Object clone() {
  try {
    // Сделать поверхностную копию
    House houseClone = (House)super.clone();
    // Сделать глубокую копию whenBuilt
    houseClone.whenBuilt = (java.util.Date)(whenBuilt.clone());
    return houseClone;
  }
  catch (CloneNotSupportedException ex) {
    return null;
  }
}
```

Теперь, если клонировать объект типа House в следующем коде:

```
House house1 = new House(1, 1750.50);
House house2 = (House)house1.clone();
```

то house1.whenBuilt == house2.whenBuilt будет равно false. Объекты house1 и house2 являются разными объектами типа Date.

Метод clone() можно реализовать в классе House без вызова метода clone() класса Object следующим образом:

```
public Object clone() {
  // Сделать поверхностную копию
  House houseClone = new House(id, area);
  // Сделать глубокую копию whenBuilt
  houseClone.whenBuilt = new Date();
  houseClone.getWhenBuilt().setTime(whenBuilt.getTime());
  return houseClone;
}
```

В этом случае класс House не нуждается в реализации интерфейса Cloneable, но необходимо убедиться, что все поля скопированы правильно. Использование метода clone() класса Object освобождает от ручного копирования полей данных. Метод clone() класса Object автоматически делает поверхностную копию всех полей данных.

Java предоставляет собственный метод, который делает поверхностную копию для клонирования объекта. Поскольку метод в интерфейсе является абстрактным, этот собственный метод не может быть реализован в интерфейсе. Поэтому для языка Java было решено определить и реализовать собственный метод clone() в классе Object.

Задание №1

1. Определить класс MyStack в соответствии со следующей таблицей:

MyStack	
-list: ArryList	Список для хранения элементов

+isEmpty():boolean	Возвращает true, если стек пуст
+getSize():int	Возвращает количество элементов в стеке
+peek():Object	Возвращает элемент на вершине стека, не удаляя его
+pop(): Object	Возвращает и удаляет элемент на вершине стека
+push(o: Object): void	Добавляет элемент в верхнюю часть стека

Совет по проектированию:
Класс MyStack содержит ArrayList, поэтому отношение между MyStack и ArrayList — композиция. По существу, композиция означает объявление переменной экземпляра для ссылки на объект. Этот объект называется составным. В то время как наследование моделирует отношение is-a, композиция моделирует отношения has-a. Также можно реализовать MyStack в качестве подкласса ArrayList. Однако использование композиции лучше, так как позволяет определить совершенно новый класс для моделирования стека без наследования ненужных и неподходящих методов класса ArrayList.

2. Убедитесь что класс MyStack реализован с помощью композиции. Определите новый класс MyStack, который наследуется от класса ArrayList. Нарисуйте UML-диаграмму этих двух классов, а затем реализуйте класс MyStack. Напишите клиент класса MyStack — программу, которая запрашивает у пользователя пять строк и отображает их в обратном порядке.
3. Перепишите класс MyStack для выполнения глубокой копии поля списка.