Лабораторна робота № 3

Тема: Реалізація класів та інтерфейсів

Цель работы. Изучить принципы использования наследования в классах

Одним из фундаментальных механизмов, лежащих в основе любого объектно-ориентированного языка, в том числе Java, является наследование. Наследование позволяет создать иерархию, при которой определенные характеристики могут передаваться от классов-родителей классам-наследникам. Реализуется наследование путем создания классов на основе уже существующих классов. При этом члены класса, на основе которого создается новый класс, с некоторыми оговорками, автоматически включаются в новый класс. Как правило, в создаваемый класс добавляют новые члены, как бы расширяют его. Согласно общепринятой терминологии, класс, на основе которого создается новый класс, называется суперклассом, а новый создаваемый на его основе класс называется подклассом.

Для реализации наследования в описании подкласса после имени класса указывается ключевое слово *extends* и имя суперкласса. Синтаксис описания подкласса имеет вид:

```
class A extends B{
// код класса
}
```

В языке Java, в отличие от языка C++, отсутствует множественное наследование, то есть подкласс в Java может создаваться на основе только одного суперкласса. При этом в Java, как и в C++, существует многоуровневое наследование: подкласс, в свою очередь, может быть суперклассом для другого класса. Благодаря многоуровневому наследованию можно создавать определенные цепочки связанных механизмом наследования классов.

Не все члены суперкласса наследуются в подклассе. Наследование не распространяется на закрытые члены суперкласса, т. е. те, которые объявляются с ключевым словом private. Под «наследование не распространяется» подразумевается невозможность в классе-наследнике создать прямой механизм использования и управления закрытыми членами класса. Т.е. экземпляры класса-наследника будут иметь закрытые поля суперкласса, но оперировать с ними будет возможно только через доступные методы суперкласса. Если таких методов нет, то и оперировать с полями будет невозможно, несмотря на их присутствие в самом объекте.

Причина такого поведения кроется в способе создания объектов подкласса. При создании объекта подкласса сначала вызывается конструктор суперкласса, а затем непосредственно конструктор подкласса. Конструктором суперкласса выделяется в памяти место для всех членов объекта, в том числе и ненаследуемых.

Если суперкласс и подкласс используют конструкторы по умолчанию, то есть ни в суперклассе, ни в подклассе конструкторы явно не описаны, то процесс создания объекта подкласса или суперкласса, с точки зрения программиста, не различаются. Если конструктору суперкласса необходимо передавать аргументы, то в конструкторе подкласса необходимо предусмотреть такую передачу аргументов, так как при создании объекта подкласса сначала автоматически вызывается конструктор суперкласса. Технически решение проблемы сводится к тому, что в программный код конструктора подкласса добавляется инструкция вызова конструктора суперкласса с указанием аргументов, которые ему передаются. Для этого используется ключевое слово super, после которого в круглых скобках указываются аргументы, передаваемые конструктору суперкласса. Инструкция вызова конструктора суперкласса указывается первой командой в теле конструктора подкласса. Таким образом, общий синтаксис объявления

конструктора подкласса имеет вид:

```
конструктор_подкласса(аргументы1){
super(аргументы2); // аргументы конструктора суперкласса
// тело конструктора подкласса
}
```

Если в теле конструктора подкласса инструкцию super не указывать, то это интерпретируется как вызов в качестве конструктора суперкласса конструктора без аргументов, т. е. такой конструктор должен быть среди конструкторов супер класса или супер класс должен использовать только конструктор по умолчанию. Во всех остальных случаях в конструкторе подкласса должно быть обращение к существующему конструктору суперкласса.

Использование механизма наследования может приводить к различным неоднозначным ситуациям, например, в подкласса и суперклассе есть поля с одним и тем же именем. С формальной и технической точек зрения в объекте такого подкласса образуются одноименные поля собственного подкласса и полученные в результате наследования. Обращение в подклассе к таким полям существенно различается - по умолчанию, обращение, выполненное в формате простого указания имени поля, использует то поле, которое описано непосредственно в подклассе, а обращение к одноименному полю суперкласса осуществляется через специальное слово super. Например, field=10; означает присвоение значения полю field, объявленного в подклассе, а super.field=100; означает присвоение значения полю field, объявленного в суперклассе. Описанная ситуация возникает, если поле field суперкласса объявлено с модификатором public или protected. Если это поле объявлено с модификатором private, то оно становится не доступным, а использование для него конструкции super.field — невозможным. Если в подклассе нет обращения к одноименному полю суперкласса, то получить доступ к этому полю суперкласса через объект подкласса, становится невозможным даже если это поле использует модификатор доступа public. Это поле становится «затененным» одноименным полем подкласса. Слово super с объектами не применяется.

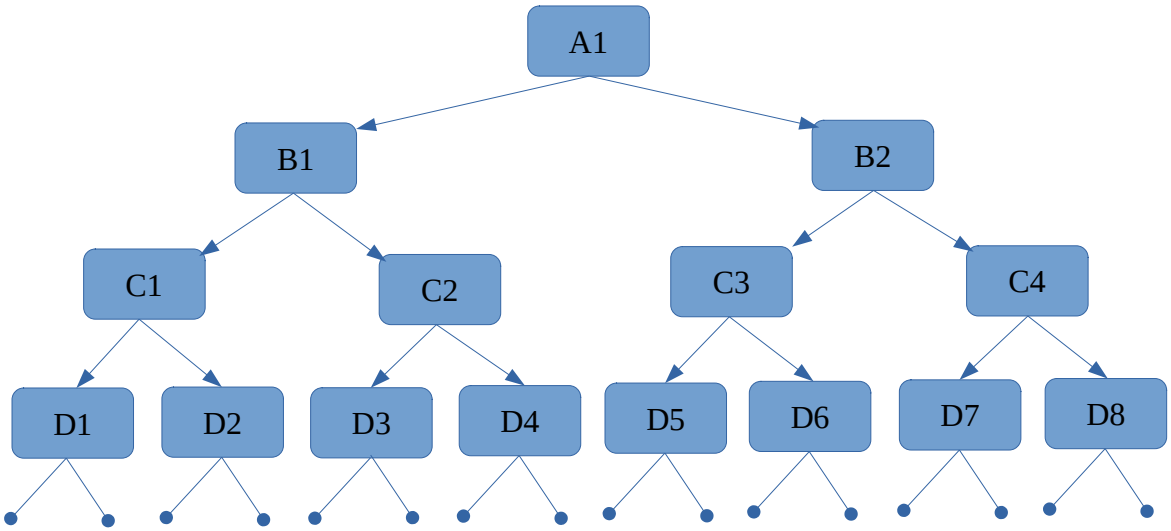
Если в подклассе описан метод с сигнатурой, совпадающей с сигнатурой метода, наследуемого из суперкласса, то метод подкласса замещает (переопределяет) метод суперкласса. Если вызывается соответствующий метод, то используется та его версия, которая описана непосредственно в подклассе. Метод из суперкласса становится доступным в подклассе, если к нему обратиться в формате ссылки с использованием ключевого слова super. Между переопределением и перегрузкой методов существует принципиальное различие - при перегрузке методы имеют одинаковые названия, но разные сигнатуры, а при переопределении совпадают не только названия методов, но и полностью сигнатуры (тип результата, имя и список аргументов). Переопределение методов реализуется только при наследовании. Перегрузка методов может быть реализована как в рамках одного класса, так и с использованием механизма наследования. Если наследуется перегруженный метод, то переопределение выполняется для каждой его версии в отдельности, причем переопределяются только те версии перегруженного метода, которые описаны в подклассе. Если в подклассе какая-то версия перегруженного метода не описана, эта версия наследуется из суперкласса. Если в суперклассе определен некий метод, а в подклассе определяется метод с таким же именем, но другой сигнатурой, то в подклассе будут доступны обе версии метода: исходная версия, описанная в суперклассе, и версия метода, описанная в подклассе. Таким образом, перегрузка метода может быть реализована через наследование.

Задания к работе

Задание 1

Полное бинарное дерево и класс его реализации

Ниже представлена программа, в которой на основе конструкторов класса создается полное бинарное дерево объектов — каждый объект имеет по две ссылки на объекты того же класса. Каждый объект имеет три поля. Символьное (типа char) поле Level определяет уровень объекта: например, в вершине иерархии находится объект уровня A, который ссылается на два объекта уровня B, которые, в свою очередь, ссылаются на четыре объекта уровня C и т. д. Объекты нумеруются, для чего используется целочисленное поле Number. Нумерация выполняется в пределах одного уровня. Например, на верхнем уровне A всего один объект с номером 1. На втором уровне B два объекта с номерами 1 и 2. На третьем уровне C четыре объекта с номерами от 1 до 4 включительно и т.д. Описанная структура объектов представлена на следующем рис.:



Кроме метки уровня и номера объекта на уровне, каждый объект еще имеет свой «идентификационный код». Этот код генерируется случайным образом при создании объекта и состоит по-умолчанию из восьми цифр (количество цифр в коде определяется закрытым статическим целочисленным полем IDnum). Код записывается в целочисленный массив, на который ссылается переменная массива ID — закрытое поле класса ObjectTree. Для вывода кода объекта используется закрытый метод showID(). Методом последовательно выводятся на экран элементы массива ID, при этом в качестве разделителя используется вертикальная черта. Сам метод вызывается в методе show(), который предназначен для отображения параметров объекта: уровня объекта в структуре, порядкового номера объекта на уровне и идентификационного кода объекта. Открытые поля FirstRef и SecondRef являются объектными переменными класса ObjectTree и предназначены для записи ссылок на объекты следующего уровня. Присваивание значений этим переменным выполняется при вызове конструктора класса.

```
package object_tree;

class ObjectTree{
```

```
private static int IDnum=8; // Количество цифр в ID-коде объекта
private char Level;         // Уровень объекта обозначенный буквой
private int Number;         // Номер объекта на определенном уровне
private int[] ID;   // Массив, который содержит ID-код объекта

ObjectTree FirstRef;        // Левый сын
ObjectTree SecondRef;       // Правый сын

private void getID(){       // Метод для генерации ID-кода
       ID=new int[IDnum];
       for(int i=0;i<IDnum;i++)
              ID[i]=(int)(Math.random()*10);
}

private void showID(){      // Метод для отображения ID-кода
       for(int i=0;i<IDnum;i++)
              System.out.print("|"+ID[i]);
              System.out.print("|\n");
}

void show(){ // Метод для отображения положения узла
       System.out.println("Уровень объекта: \t"+Level);
       System.out.println("Номер на уровне: \t"+Number);
       System.out.print("ID-код объекта: \t");
       showID();
}

ObjectTree(int k,char L,int n){  // Конструктор бинарного дерева: k –
                                 // количество уровней дерева, L – буквенное
                                 // обозначение уровня, n – номер узла на уровне
       System.out.println("\tСоздан новый объект!");
       Level=L;
       Number=n;
       getID();
       show();
       if(k==1){
              FirstRef=null;
              SecondRef=null;
       }
       else{
              // Рекурсивный вызов конструктора для левого и правого сына
              FirstRef=new ObjectTree(k-1,(char)((int)L+1),2*n-1);
              SecondRef=new ObjectTree(k-1,(char)((int)L+1),2*n);}
       }
}

class ObjectTreeDemo{

       public static void main(String[] args){

              ObjectTree tree=new ObjectTree(4,'A',1); // Создание дерева
              System.out.println("\tПроверка структуры дерева объектов!");
              tree.FirstRef.SecondRef.show();// Вывод объекта С 2
       }
}
```

Представленный выше класс служит только примером построения бинарного дерева и не имеет прямого практического значения.

Для улучшения практической значимости представленного класса дополните его полем value (для примера задайте ему тип int), а также:

- методом, который позволяет определить полноту дерева (т. е. наличие левого и правого сына для всех объектов (узлов), кроме листьев и возможно последнего узла-объекта, у которого может быть только левый сын);
- методом, который проверяет возможность добавления узла по указанному адресу (уровень-номер);
- методом, который добавляет узел по указанному адресу;
- методом, который удаляет узел по указанному адресу;

- методом, который восстанавливает полноту двоичного дерева после удаления узла;
- методом, который позволяет объединить два дерева.

При необходимости измените модификаторы доступа уже имеющихся методов и полей.

Задание 2

Дополните класс полного бинарного дерева конструктором, который позволяет построить структуру с указанным количеством объектов-узлов.

Задание 3

Разработать класс Пирамида, который является наследником класса полного бинарного дерева. Переопределите метод восстановления полноты бинарного дерева после удаления узла, на метод восстановления главного свойства Пирамиды — у каждого из сыновей значение value больше, чем у отца. Переопределите конструкторы так, чтобы они позволяли строить Пирамиду с поддержкой ее главного свойства. Разработайте конструктор, который позволяет построить Пирамиду на основе одномерного массива значений для его объектов.

Программный код и результаты работы (с возможными пояснениями) представьте в отчете.