Державний університет телекомунікацій
Навчально-науковий інститут телекомунікацій та інформатизації
Кафедра комутаційних систем

Методичне керівництво для виконання лабораторної роботи №3

Призначення класів, методів, конструктори. Типи відносин між класами.

з дисципліни «Програмне забезпечення телекомунікаційних систем».
освітньо-кваліфікаційного рівня магістр

Київ - 2014

Укладачі: проф. каф. КС Кунах Н.І.
доц. каф. КС Невдачина О.В.
ст.викладач КС Полоневич А.П.

Методичні вказівки обговорені і схвалені
на засіданні кафедри КС

Протокол № _____
від «___» _____ 2014р.

Тема: Класи, методи, конструктори. Типи відносин між класами. Модифікатори доступу.

Мета заняття:
Навчитися створювати основні структурні еллементи програми мовою програмування Java.

Час проведення заняття: 90 хвилин.

Список літератури.

1. Гребешков А.Ю. «Микропроцессорные системы и программное обеспечение в средствах связи», ПГУТИ Самара, 2009. (надається в електронному вигляді).
2. Голицина О.Л., Партыка Т.Л. «Програмное обеспечение», 2-ое издание, Москва, 2008. (надається в електронному вигляді).

Зміст заняття.

1. **Ознайомча частина.**

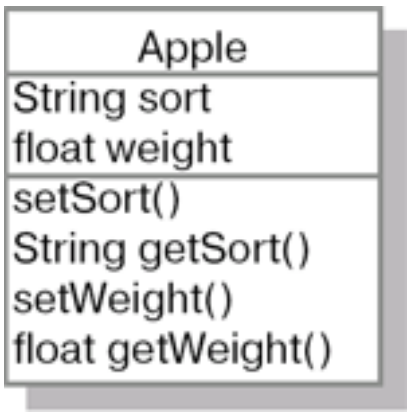
Перевірка присутності студентів, оголошення теми заняття та порядок його проведення.

2. **Матеріал для вивчення:**

Найбільш важливою властивістю класу є те, що він визначає новий тип данних. Після того як він визначений, цей новий тип можливо використовувати для створення нових об`єктів даного типу. Таким чином, клас-це шаблон об'єкта, а об'єкт-це екземпляр класу. Термін екземпляр та об'єкт це синоніми.

Клас - це шаблон поведінки об'єктів певного типу із заданими параметрами, що визначають стан. Усі примірники одного класу (об'єкти, породжені від одного класу) мають один і той же набір властивостей і загальна поведінка, тобто однаково реагують на однакові повідомлення.

Відповідно до UML (Unified Modelling Language - уніфікована мова моделювання), клас має наступне графічне представлення:



Клас зображується у вигляді прямокутника, що складається з трьох частин. У верхній частині поміщається назва класу, в середній - властивості об'єктів класу (атрибути), у нижній - дії, які можна виконувати з об'єктами даного класу (методи).

При визначенні класу оголошують його конкретну форму та сутність. Для цього вказують дані, які він вміщує, та коди, які впливають на ці дані.

Для об'явлення класа служить ключове слово ***class*.**

```
class ім'я_класу {
тип змінна_екземпляру1;
тип змінна_екземпляру2;
//...
тип змінна_екземпляруN;
тип ім'я методу1 (список_параметрів) {
//тєло методу
}
тип ім'я методуN (список_параметрів) {
//тєло методу
}
}
```

Дані, чи змінні, які визначені всередені класу, називають **змінними екземпляру**. Код міститься всередині методу.

Визначені всередені класу методи та змінні разом називають **членами класу.**

Визначені всередені класу змінні називають змінними екземпляру, оскільки кожен екземпляр класу (тобто кожен об'єкт класу) містить свої власні копії екземпляру класу. Таким чином, дані одного об'єкта відокремленні та відрізняються від даних іншого об'єкта.

Всі методи мають ту саму загальну форму, що й метод main( ). Однак, більшість методів не вказують як static чи pablic. Зверніть увагу, що загальна форма класу не містить визначення методу main (). Класи в Java можуть не містити цей метод. Його обов'язково зазначати лише у тих випадках, коли даний клас слугує початковою точкою програми.

Простий клас.

У прикладі приведено код класу, який містить три змінних екземпляру: width (ширина), height (висота), depth (глибина). В даний момент клас Box не вміщує ніяких методів.

В даному прикладі, клас визначає тип даних Bax. Об'явлення Class створює тільки шаблон, але не дійсний об'єкт.

Щоб дійсно створити об'єкт класу Box, потрібно використовувати оператор, наприклад, наступний.

Box mybox = new Box(); //створення об'єкту mybox класу Box.

Після виконання цієї операції mybox стане екземпляром класу Box. Тобто він отримує "фізичне існування".

Кожен об'єкт класу Box буде містити свої копії змінних екземпляру width, height, depth. Для доступу до цих змінних використовується оператор крапка(.). Цей оператор зєднує ім'я об'єкта з ім'ям змінної екземпляру

Наприклад, щоб присвоїти змінній widht об'єкту mybox значення 100, потрібно використати наступний оператор:

mybox.widht=100;

Цей оператор вказує компілятору, що копії змінної widht, яка зберігається всередені об'єкту mybox потрібно присвоїти значення 100.

Приклад: повна програма, в якій використовується клас Box.

Файлу цієї програми слід присвоїти ім'я BoxDemo.java, оскільки метод main() визначений в клас, який має назву BoxDemo, а не Box. Після компіляції ви отримаєте два файли .class: для BoxDemo та Box.

Після запуску програми Ви отримаєте наступний результат: Об'єм дорівнює 3000.0

Кожен об'єкт вміщує власні копії змінних екземпляру. Це означає, що при наявності двух об'єктів класу Box кожен з них бути вміщувати власні копії змінних width, height, depth. Важливо, що зміна змінних екземпляра одного об'єкта не впливають на змінні екземпляра іншого. В наступній програмі створюється два об'єкти.

Ця програма створює наступний вивод.

Об'єм дорівнює 3000.0
Об'єм дорівнює 162.0

Виклик метода volume () виконується у правій частині оператора присвоювання. права частина цього оператора є змінна, у даному випадку vol, яка буде приймати значення, яке повертатиме метод volume (). Таким чином, після виконання оператора
vol=mybox1.volume ();
метод mybox1. volume () поверне значення 3000, и цей об'єм зберігається у змінній vol. При роботі зі значеннями що повертаються слід враховувати дві важливі обставини.

1. Тип даних, який повертається методом, повинен бути сумісним з типом що повертається, указаним методом.

2. Змінна, яка приймає повертаєме методом значення, також повинна бути сумісною з типом, що повертається, що вказаний для метода.

Типи відносин між класами

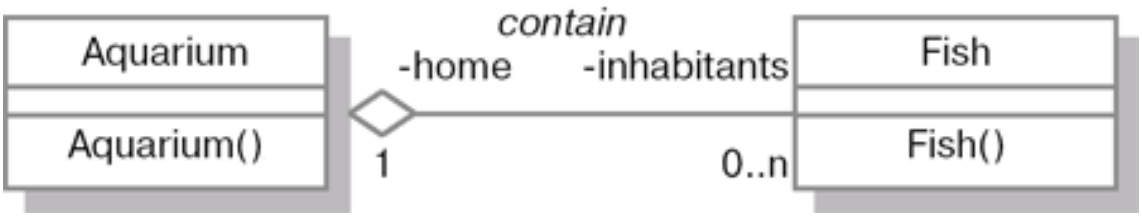
Як правило, будь-яка програма, написана на Java являє собою деякий набір пов'язаних між собою класів. Можна провести аналогію між написанням програми і будівництвом будинку. Подібно до того, як стіна складається з цеглин, комп'ютерна програма з використанням Java будується з класів. Причому ці класи повинні мати уявлення один про одного, для того щоб спільно виконувати поставлене завдання.

Можливі такі зв'язки між класами в рамках об'єктної моделі (наводяться лише найбільш прості і часто використовувані види зв'язків):

- агрегація (Aggregation);
- *ассоціація* (Association);
- *наслідування* (Inheritance);
- *метакласи* (*Metaclass*).

Агрегація

Відношення між класами типу "містить" (contain) або "складається з" називається агрегацією, або включенням. Наприклад, якщо акваріум наповнений водою і в ньому плавають рибки, то можна сказати, що акваріум агрегує в собі воду і рибок.



Таке ставлення включення, або агрегації (aggregation), зображується лінією з ромбиком на боці того класу, який виступає в якості власника, або контейнера. Необов'язкове назву відносини записується посередині лінії.

У нашому прикладі ставлення contain є двонаправленим. Об'єкт класу Aquarium містить кілька об'єктів Fish. У той же час кожна рибка "знає", в якому саме акваріумі вона живе. Кожен клас має свою роль в агрегації, яка вказує, яке місце займає клас в даному відношенні. Ім'я ролі не є обов'язковим елементом позначень і може бути відсутнім на діаграмі. У прикладі можна бачити роль home класу Aquarium (акваріум є домом для рибок), а також роль inhabitants класу Fish (рибки є мешканцями акваріума). Назва ролі зазвичай збігається з назвою відповідного поля в класі. Зображення такого поля на діаграмі зайве, якщо вже вказано ім'я ролі. Тобто в даному випадку клас Aquarium матиме властивість (поле) inhabitants, а клас Fish - властивість home.

Число об'єктів, що беруть участь у відношенні, записується поряд з ім'ям ролі. Запис "0 .. n" означає "від нуля до нескінченності". Прийнято також позначення:

- " 1..n " - від одиниці до нескінченності;
- " 0 " - нуль;
- " 1 " - один;
- " n " - фіксована кількість;
- " 0..1 " - нуль або один.

Код, що описує розглянуту модель і явище агрегації, може виглядати, наприклад, наступним чином:

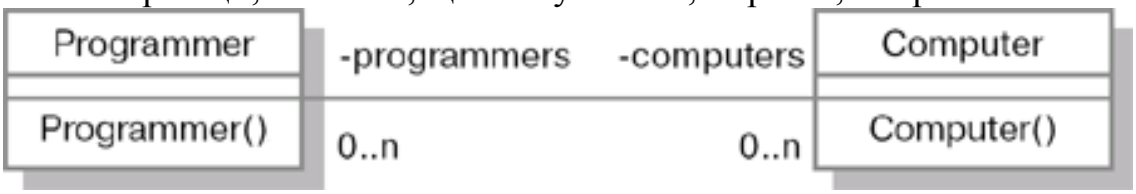
```
// определение класса Fish
public class Fish {
  // определения поля home
  // (ссылка на объект Aquarium)
  private Aquarium home;

  public Fish() {
  }
}
// определение класса Aquarium
public class Aquarium {
  // определения поля inhabitants
  // (массив ссылок на объекты Fish)
  private Fish inhabitants[];
```

```
  public Aquarium() {
  }
}
```

Асоціація

Якщо об'єкти одного класу посилаються на один або більше об'єктів іншого класу, але ні в ту, ні в іншу сторону відношення між об'єктами не носить характеру "володіння", або контейнеризації, таке ставлення називають асоціацією (association). Ставлення асоціації зображується так само, як і ставлення агрегації, але лінія, що зв'язує класи, - проста, без ромбика.

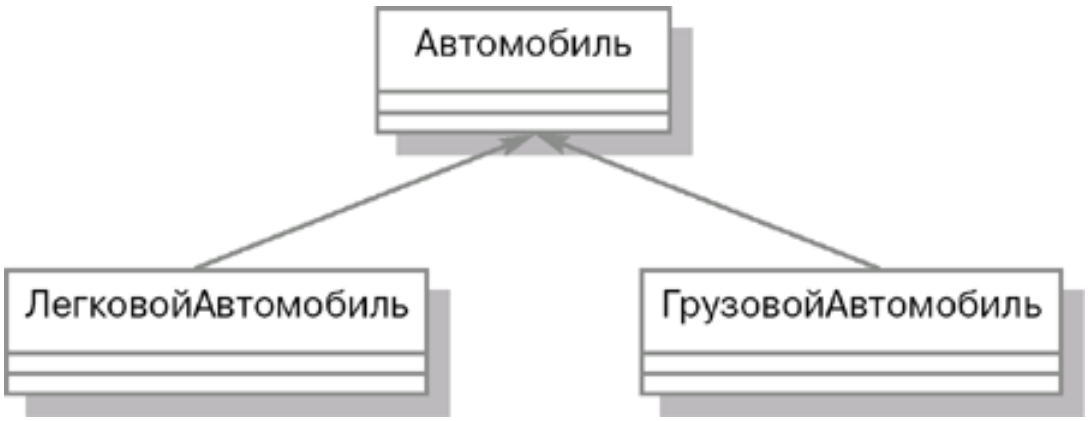


У даному випадку між екземплярами класів Programmer і Computer в обидві сторони використовується відношення "0 .. n", тому що програміст, в принципі, може не працювати з комп'ютером (якщо він теоретик або на пенсії). У свою чергу, комп'ютер може ніким не використовуватися (якщо він новий і ще не встановлений).
Код, відповідний розглянутого прикладу, буде, наприклад, наступним:

```
public class Programmer {
  private Computer computers[];
  public Programmer() {
  }
}
public class Computer {
  private Programmer programmers[];
  public Computer() {
  }
}
```

Спадкування

Спадкування (inheritance) - це відношення між класами, при якому клас використовує структуру або поведінку іншого класу (одиночне спадкоємство), або інших (множинне спадкоємство) класів. Спадкування вводить ієрархію "загальне / приватне", в якій підклас успадковує від одного або декількох більш загальних суперкласів. Підкласи зазвичай доповнюють або перевизначають успадковану структуру і поведінку.



Ставлення узагальнення позначається суцільною лінією з трикутною стрілкою на кінці. Стрілка вказує на більш загальний клас (клас-предок або суперклас), а її відсутність - на більш спеціальний клас (клас-нащадок або підклас).

Використання успадкування сприяє зменшенню кількості коду, створеного для опису схожих сутностей, а також сприяє написанню ефективнішого і гнучкого коду.

Модифікатори доступу

Розмежування доступу в Java

Рівень доступу елемента мови є статичним властивістю, задається на рівні коду і завжди перевіряється під час компіляції. Спроба звернутися до закритого елементу безпосередньо викличе помилку.

У Java модифікатори доступу вказуються для:

• типів (класів та інтерфейсів) оголошення верхнього рівня;
• елементів посилальних типів (полів, методів, внутрішніх типів);
• конструкторів класів.

Як наслідок, масив також може бути недоступним у тому випадку, якщо недоступний тип, на основі якого він оголошений. Всі чотири рівня доступу мають тільки елементи типів і конструктори. це:

- public;
- private;
- protected;
- якщо не вказаний жоден з цих трьох типів, то рівень доступу визначається за замовчуванням (default).

У багатьох мовах існують права доступу, які обмежують можливість використання, наприклад, змінної в класі. Наприклад, легко уявити два крайніх виду прав доступу: це public, коли поле доступно з будь-якої точки

програми, і private, коли поле може використовуватися тільки усередині того класу, в якому воно оголошено.

Останній рівень (доступ за умовчанням) - він допускає звернення з того ж пакета, де оголошений і сам цей клас. З цієї причини пакети в Java є не просто набором типів, а більш структурованою одиницею, так як типи всередині одного пакета можуть більше взаємодіяти один з одним, ніж з типами з інших пакетів.

Нарешті, protected дає доступ спадкоємцям класу. Зрозуміло, що спадкоємцям може знадобитися доступ до деяких елементів батька, з якими не доводиться мати справу зовнішнім класам.

Однак описана структура не дозволяє впорядкувати модифікатори доступу так, щоб кожен наступний строго розширював попередній. Модифікатор protected може бути вказаний для спадкоємця з іншого пакета, а доступ за замовчуванням допускає звернення з класів-ненаслідників, якщо вони знаходяться в тому ж пакеті. З цієї причини можливості protected були розширені таким чином, що він включає в себе доступ всередині пакета. Отже, модифікатори доступу упорядковуються наступним чином (від менш відкритих - до більш відкритим):

private
(none) default
protected
public

Модифікатори доступу можливі для різних елементів мови.

- Пакети доступні завжди, тому у них немає модифікаторів доступу (можна сказати, що всі вони public, тобто будь-який існуючий в системі пакет може використовуватися з будь-якої точки програми).
- Типи (класи та інтерфейси) верхнього рівня оголошення. При їх оголошенні існує всього дві можливості: вказати модифікатор public або не вказувати його. Якщо доступ до типу є public, то це означає, що він доступний з будь-якої точки коду. Якщо ж він не public, то рівень доступу призначається за умовчанням: тип доступний тільки всередині того пакету, де він оголошений.
- Масив має той же рівень доступу, що й тип, на основі якого він оголошений (природно, все примітивні типи є повністю доступними).
- Елементи й конструктори об'єктних типів. Володіють всіма чотирма можливими значеннями рівня доступу. Всі елементи інтерфейсів є public.

Для типів оголошення верхнього рівня немає необхідності у всіх чотирьох рівнях доступу. Private-типи утворювали б закриту міні-програму, ніхто не міг би їх використовувати. Типи, доступні тільки для спадкоємців, також не були визнані корисними.

Розмежування доступу позначаються не тільки на зверненні до елементів об'єктних типів або пакетів (через складене ім'я або пряме

звернення), але також при виклику конструкторів, спадкуванні, приведенні типів. Імпортувати недоступні типи забороняється.

Перевірка рівня доступу проводиться компілятором. Зверніть увагу на наступні приклади:

```
public class Wheel {
  private double radius;
  public double getRadius() {
    return radius;
  }
}
```

Значення поля radius недоступно зовні класу, однак відкритий метод getRadius () коректно повертає його.

Розглянемо наступні два модулі компіляції:

```
package first;

// Некоторый класс Parent
public class Parent {
}
```

```
package first;

// Класс Child наследуется от класса Parent,
// но имеет ограничение доступа по умолчанию
class Child extends Parent {
}

public class Provider {
  public Parent getValue() {
    return new Child();
  }
}
```

До методу getValue () класу Provider можна звернутися і з іншого пакета, не тільки з пакету first, оскільки метод оголошений як public. Даний метод повертає екземпляр класу Child, який недоступний з інших пакетів. Проте наступний виклик є коректним:

```
package second;

import first.*;

public class Test {
 public static void main(String s[])
 {
  Provider pr = new Provider();
  Parent p = pr.getValue();
  System.out.println(p.getClass().getName());
```

```
  // (Child)p - приведет к ошибке компиляции!
 }
}
```

Результатом буде:

```
first.Child
```

Тобто насправді в класі Test робота йде з примірником недоступного класу Child, що можливо, оскільки звернення до нього робиться через відкритий клас Parent. Спроба ж виконати явне приведення викличе помилку. Так, тип об'єкта "вгаданий" вірно, але доступ до закритого типу завжди заборонений.

Наступний приклад:

```
          public class Point {
 private int x, y;

 public boolean equals(Object o) {
  if (o instanceof Point) {
   Point p = (Point)o;
   return p.x==x && p.y==y;
  }
  return false;
 }
}
```

У цьому прикладі оголошується клас Point з двома полями, що описують координати точки. Зверніть увагу, що поля повністю закриті - private. Далі спробуємо перевизначити стандартний метод equals () таким чином, щоб для аргументів, які є екземплярами класу Point, або його спадкоємців (логіка роботи оператора instanceof), в разі рівності координат поверталося справжнє значення. Зверніть увагу на рядок, де робиться порівняння координат, - для цього доводиться звертатися до private-полях іншого об'єкта!

Проте, така дія коректно, оскільки private допускає звернення з будь-якої точки класу, незалежно від того, до якого саме об'єкту воно проводиться.

3. Питання для самоконтролю.

1. Що таке Клас?
2. Які відносини між класами Ви знаєте?
3. Що таке "агрегація"?
4. Що таке "асоціація"?
5. Що таке "наслідування" класів?
6. З якою метою у мові програмування Java створене наслідування класів?

7. Для чого у Java вказуються модифікатори доступу?
8. Які модифікатори доступу Вам відомі?