Увод

в обектно ориентираното програмиране

ООП – Преговор

Съдържание

- ООП - основи
- Основни принципи
- Клас
- Какво има в един клас
- Нива на достъп
- Наследяване

Какво е ООП?

Какво е то?

- Парадигма начин на структуриране на кода чрез класове и обекти
- ООП моделира обектите от реалния свят и взаимоотношенията между тях

Стол

- Цвят
- Материал
- Брой крака
- Позиция
- Може да бъде местен
- Празен или зает

Защо го използваме?

- Добро структуриране на кода
- Намалява сложността на кода
- Позволява преизползване на кода
- Може да се постигне абстрактност

ООП

- Основна единица на обектно ориентираното програмиране е класа.
- ООП започва от идентифицирането на класовете и преминава в имплементиране на методите в тях.
- Класът дефинира променливите и методите, които обектите ще имат.
- Обект е представител на клас. Всеки обект е от даден клас, който му дефинира свойствата и възможностите.

Overriding

Когато един клас наследява друг:

Наследника може да промени поведението на някои от методите на базовия клас. Той ги предефинира.

```
class Human {

   public void eat() {

   sysout(“eating loudly”);

   }

}
```

```
class Lady extends Human {

    @Override public void eat() {

    sysout(“eating politely”);

    }

}
```

Overriding

@Override не е задължително, но така сте сигурни, че предефинирате нещо, компилатора ще ви предупреди, ако не е така

Не можете да предефинирате final или static метод

Не можете да предефинирате конструктор

Overloading

Този термин се използва, когато съществуват два метода с едно и също име в един клас. Компилатора при създаване на вашата програма избира кой да използва спрямо вида и броя на параметрите.

```
class Human {
   public void eat() {
   sysout(“eating louduly”);
   }
   public void eat(int n) {
   sysout(“eating louduly for ” + n + “ hours”);
   }
}
```

```
class Test {
   ..main..
   Human jeff = new Human();
   jeff.eat(); jeff.eat(6);
}
```

Основни принципи

Основни принципи

- Капсулация: знае се КАКВО може да прави компонента, а не как
- Наследяване: един обект със същите свойства като друг може да го наследи
- Полиморфизъм: позволява унифицирано извършване на действия над различни обекти

Стол

```
class Chair {

String material = “wood”;

int positionX = 5;

int positionY = 4;

Color matColor = Color.RED;

void moveChair ( int x, int y){

…

}
```

Polymorphism

- one name, many forms
- Да имаш много методи с едно и също име, но с еко различно поведение
- Постига се чрез overridding, наричан run-time polymorphism, и overloading, наричан compile-time polymorphism

Overriding

Когато един клас наследява друг:

Наследника може да промени поведението на някои от методите на базовия клас. Той ги предефинира.

```
class Human {

   public void eat() {

   sysout(“eating loudly”);

   }

}
```

```
class Lady extends Human {

    @Override public void eat() {

    sysout(“eating politely”);

    }

}
```

Overriding

@Override не е задължително, но така сте сигурни, че предефинирате нещо, компилатора ще ви предупреди, ако не е така

Не можете да предефинирате final или static метод

Не можете да предефинирате конструктор

Overloading

Този термин се използва, когато съществуват два метода с едно и също име в един клас. Компилатора при създаване на вашата програма избира кой да използва спрямо вида и броя на параметрите.

```
class Human {
   public void eat() {
   sysout(“eating louduly”);
   }
   public void eat(int n) {
   sysout(“eating louduly for ” + n + “ hours”);
   }
}
```

```
class Test {
   ..main..
   Human jeff = new Human();
   jeff.eat(); jeff.eat(6);
}
```

Polymorphism

- one name, many forms
- Да имаш много методи с едно и също име, но с еко различно поведение
- Постига се чрез overridding, наричан run-time polymorphism, и overloading, наричан compile-time polymorphism

Интерфейси

Дефинират списък от операции, методи, без да дефинират самите тях

Нещо като обещание, че един клас ще има дадени методи

Дефинират абстрактни типове данни

```
class Dog implements IBarkable {
   public void bark() {
   sysout(“bau”);
   }
}
```

```
interface IBarkable{
void bark();
}
```

Абстракция

- Създаването на класове, обекти и типове по техните интерфейси и функционалност вместо по имплементационните им детайли
- Възможността да взаимодействаш не само с конкретен клас, а с всички класове правещи дадено нещо

Статичност

Дадено поле или метод се асоциират с класа, а не с обект от него

Съществува само едно копие от тази променлива и то се използва от всички обекти от класа.

Тези атрибути се извикват с името на класа

Инициализират се при първото използване на този клас

Клас

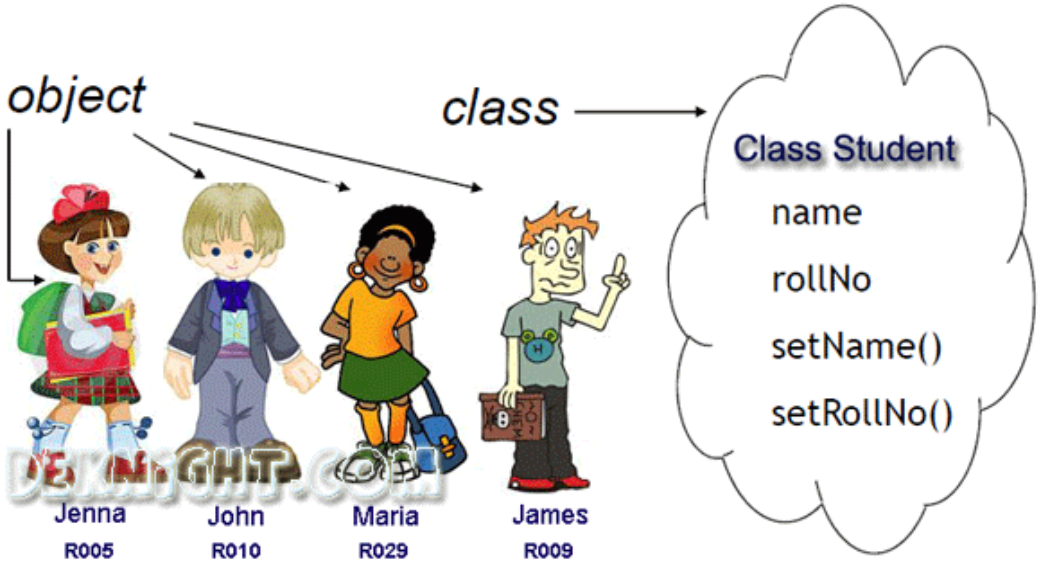
Клас

- Класът е шаблон от който създаваме обект
- Обекта е конкретен елемент от даден клас. Нарича се още инстанция на класа

Student pesho = new Student();

Chair zelen = new Chair();

zelen.color = Color.Green;



Капсулация

- Защита на данните и имплементацията
- Скриването на имплементацията на данните чрез ограничаване на достъпа до мутаторите
- Можем да правим промени на обекта без да се тревожим, че ще счупим другия код, който извиква методите от класа за информация

Какво има в един клас

Какво има в един клас

- Полета:
- String name;
- int age;
- Методи:
- void study(){…}
- void doHomework(){…}
- Конструктор
- Други неща

Конструктор

- Конструктора е метод, който се извиква автоматично при създаването на обект от класа и само тогава
- Използва се за да се зададат първоначални стойности на полетата и за да се направят първоначални настройки на класа
- Един клас може да има много конструктори

Нива на достъп

Нива на достъп

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
no modifier	✓	✓	✗	✗
private	✓	✗	✗	✗

GETTER и SETTER

GETTER и SETTER

- Ограничават достъпа до полетата.
- Предотвратяват унищожаване на данни
- Или подмяната им (още по-лошо!!!)

```
private int age;
public getAge() {
return age;
}
public setAge(int newAge) {
age = newAge;
}
```

Оператора this

- Позволява обръщане към обекта вътре от самия обект

Все едно правим

String text = “asd”;

text.substring(5);

но от вътрешността на класа String

this.substring(5) – това ще се обърне към сегашния обект

Наследяване

Наследяване

Класът-дете получава всички методи и полета на класа-родител

```
class Human {
  int age;
  void getOld(){
  this.age++;
  }
}
```

```
class Student extends Human {
  int number;
  void doHW(){
  }
}
```

Наследяване

- Една от най-силните черти на наследството е възможността за extend-ване на компоненти без да се знае нищо за начина, по който са имплеметирани в базовия клас
- Обектите могат да бъдат свързани помежду си чрез връзка от типа “има”, „използва“ и „е“ . Именно „е“ връзката е начина на наследяване на един обект от друг. (Когато можем да кажем, че един обек е от типа друг обект)

Пример:
Клас човек;
Класа ученик е човек;

Един клас има

- Полета: променливи, които определят настоящия статус на обекта
- Статични полета: променливи, които са общи за всички обекти от класа и по-скоро определят статуса на класа като цяло, не на отделните обекти от този клас.
- Методи: изпълним код, позволяващ ни да променяме състоянието на обекта или да достъпваме данни от него
- Статични методи: изпълним код, отнасящ се за класа като цяло, не трябва да използва в себе си променливи, които не са статични
- Вложени класове и интерфейси

Структури от данни

Структури от данни

В зависимост от задачата, която трябва да решим с програмиране, се налага да организираме данните, с които работим, по различен начин (например подредба на някакви елементи или връзки между тях.)

Структурите от данни са множество от данни, организирани по определен начин.

Абстрактен тип данни Abstract Data Type (ADT)

При разглеждането на типовете данни, се интересуваме от действията, които могат да се извършват, без да се интересуваме от начина, по който реализират.

Пример

Абстрактна дефиниция на масив

За масивите е характерно:

- Можем да достъпваме елементите по индекс и да променяме елемент в даден индекс
- Можем да вземем дължината на масива
- Не можем да променяме броя на елементите (не можем да добавяме и премахваме елемент)

Линейни структури от данни

Линейни структури от данни

Има различни структури от данни. Днес ще разгледаме линейните структури от данни. Те са най-често срещаните.

Представляват описание (абстракция) на поредица (списък) от обекти от реалния свят.

Пример

Имаме списък със задачи по даден проект. Искаме да можем да:

- Добавяме задача
- Изтриваме задача
- Достъпваме задача (за да можем да я променим, например)
- Проверяваме дали списъкът е празен
- Да сменяме местата на задачи

Структури от данни в Java

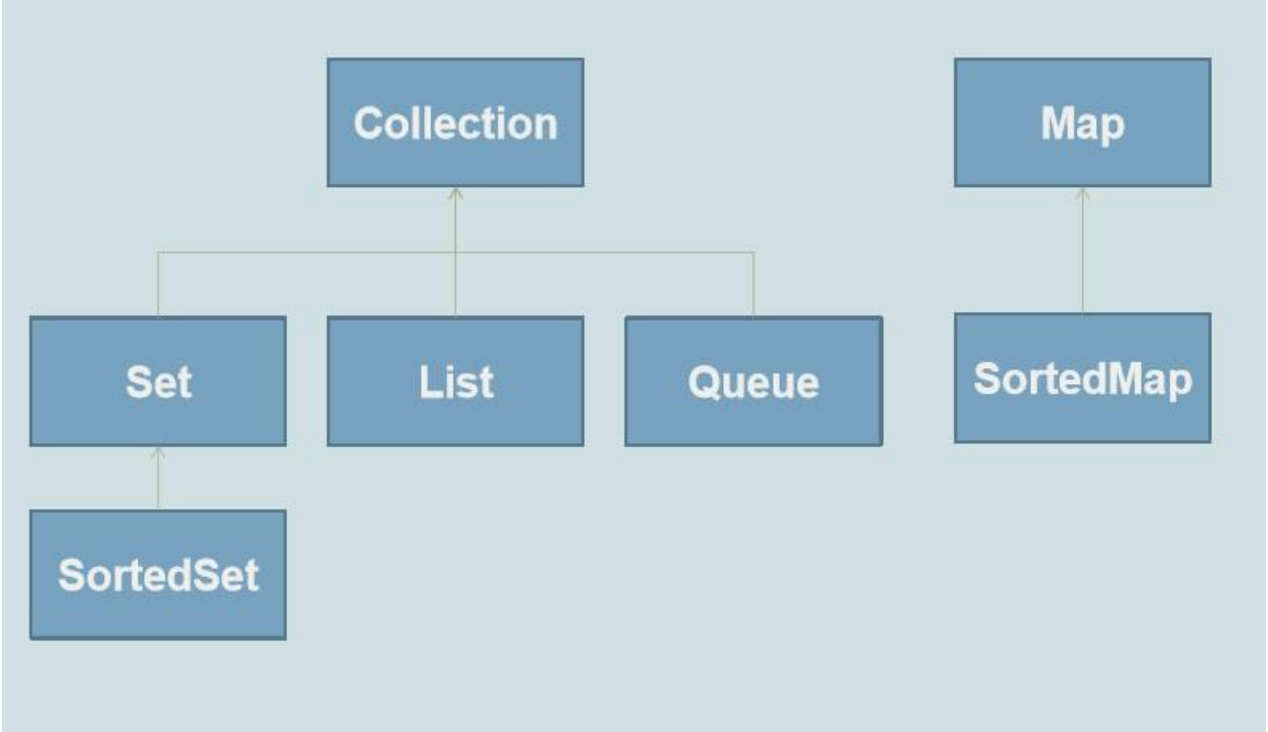
Структури от данни в Java

Колекциите са структурите от данни в стандартната библиотека на Java.

Collections Framework в Java включва:

- интерфейси
- конкретни реализации (класове) на тези интерфейси
- алгоритми

Интерфейси



Стандартни имплементации

	Hash table	Resizable Array	Tree	Linked List	Hash table + linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue				LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Алгоритми

- Събрани в класа Collections (Arrays за масиви)
- Повечето оперират върху List, а не върху Collection

Основни алгоритми

- сортиране
- разбъркване
- обръщане
- копиране
- размяна на елементи
- добавяне на всички елементи
- търсене
- намиране на най-голяма и най-малка стойност

Класове за списъци в Java

- ArrayList: Реализиран с масив
- LinkedList: Реализиран със свързан списък

По-често се използва ArrayList.

Работа с файлове

Потоци

- Последователност от байтове, които се изпращат от едно приложение към друго, или от един компютър към друг, или от приложение към файл и обратно
- Имат подредба

Потоците

- Позволяват четене и писане на данни
- Са подредени
- Достъпът при тях е последователен
- Трябва да бъдат затваряни, за да не се получи повреждане на файла

Основни класове

Основните класове са InputStream & OutputStream. Те са абстрактни. Има конкретни реализации, които ги наследяват:

- BufferedInputStream BufferedOutputStream
- DataInputStream, DataOutputStream,
- Reader, Writer,
- BufferedReader, BufferedWriter,
- PrintWriter и PrintStream

Четене от текстов файл

Ще използваме Scanner:

```
// Link the File variable to a file on the computer

File file = new File("test.txt");

// Create a Scanner connected to a file and specify encoding

Scanner fileReader = new Scanner(file, "windows-1251");

// Read file here...

// Close the resource after you've finished using it

fileReader.close();
```

Четене от текстов файл

```
int lineNumber = 0;

// Read file

while (fileReader.hasNextLine()) {

lineNumber++;

System.out.printf("Line %d: %s%n", lineNumber, fileReader.nextLine());

}
```

Писане в текстов файл

PrintStream има същите методи като при писане на конзолата:

```
// Create a PrintStream instance

PrintStream fileWriter = new PrintStream("numbers.txt");

// Loop through the numbers from 1 to 20 and write them

for (int num = 1; num <= 20; num++) {

fileWriter.println(num);

}

// Close the stream when you are done using it

fileWriter.close();
```

Важно

Не пропускайте да затворите потока след като приключите с използването му! За затваряне използвайте метода close().

Try / Catch

Когато един код има потенциала да хвърли грешка, трябва да се обгърне в try { } блок.

Той пречи на грешката да прекъсне програмата. Try трябва да бъде последван от catch {} блок. Той се изпълнява, АКО в try е хвърлена грешка. По желание след тях може да се сложи finally {} блок, той се изпълнява винаги.

Пример

```
try {

PrintStream fileWriter = new PrintStream("file.txt");

} catch (Exception e) {

System.println(“We had an error”);

} finally {

fileWriter.close();

}
```