devcraft.info

Оценяване 1/2

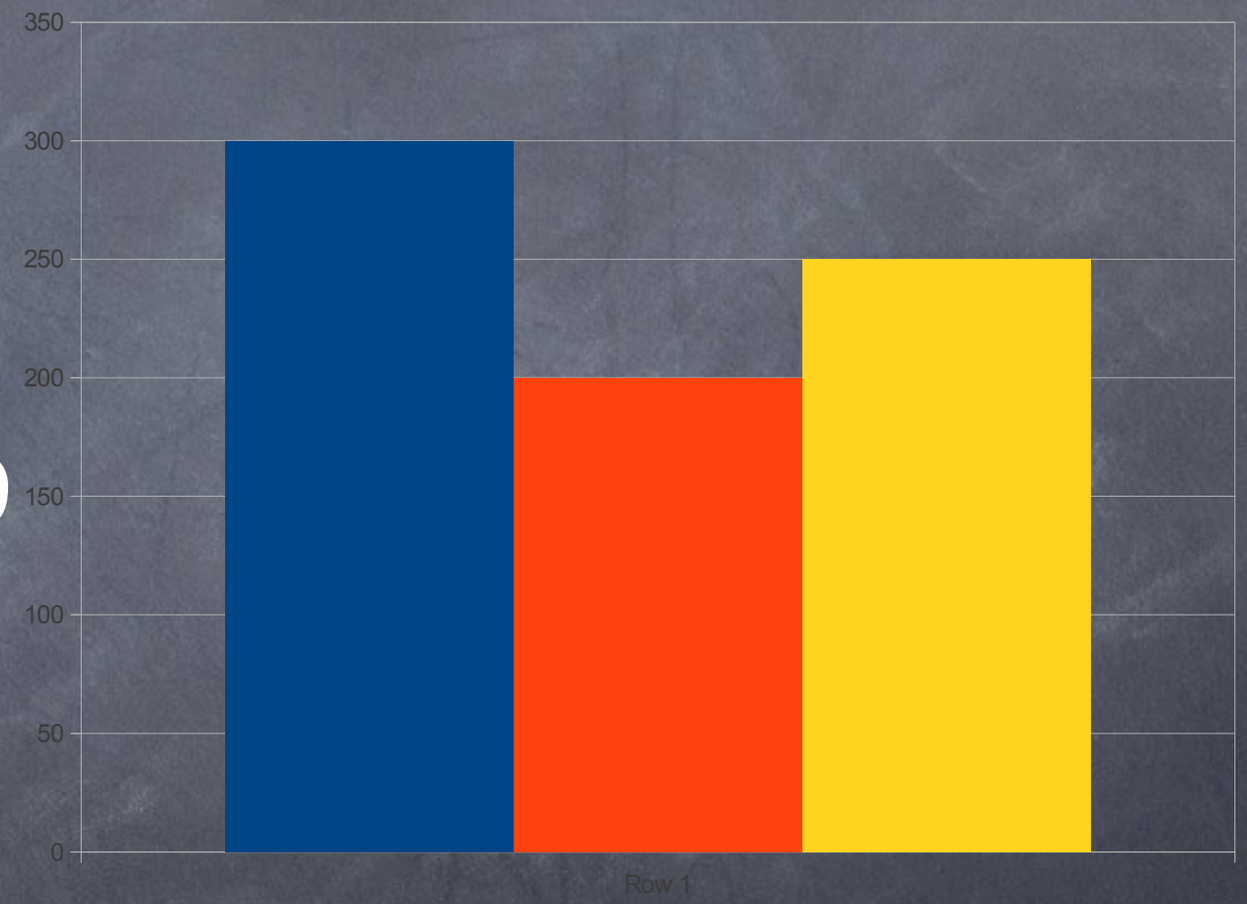
Задачи: 100 точки

Проект: 100 точки

Тест: 100 точки

Оценяване 2/2

Максимален брой: 300
Сертификат: 200
Отличен сертификат: 250



поне 50 точки от задачите

Задачите от лекция 2

Сумата на числата от 1 до 1000

```
public class Exercise1 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 1; i <= 1000; i++) {
            sum += i;
        }

        System.out.printf("The sum is: %d", sum);
    }
}
```

Задачите от лекция 2

Сумата на числата от 1 до 1000, които се делят на 3 или на 5

```
public class Exercise2 {
    public static void main(String[] args) {
        int sum = 0;
        for (int i = 0; i < 1000; i++) {
            if (i % 3 == 0 || i % 5 == 0) {
                sum += i;
            }
        }

        System.out.printf("The answer is: %d %n" + sum);
    }
}
```

Задачите от лекция 2

Сумата от цифрите на 100!

```
public class Exercise3 {
    public static void main(String[] args) {
        BigInteger factorial = BigInteger.valueOf(1);
        int digitSum = 0;

        for (int i = 2; i <= 100; i++) {
            factorial = factorial.multiply(BigInteger.valueOf(i));
        }

        String digit = factorial.toString();

        for (int i = 0; i < digit.length(); i++){
            Character a =  digit.charAt(i);
            digitSum += Integer.parseInt(a.toString());
        }

        System.out.printf("The sum of digits is: %d %n", digitSum);
    }
}
```

Ruby Magic

```
(1..1000).reduce(:+)
```

```
(1..1000).select do |number|
  number % 3 == 0 || number % 5 == 0
end.reduce(:+)
```

```
(1..100).reduce(:*).to_s.split(//).inject(0) do |r, c|
  r + c.to_i
end
```

Пролет 2012

Задачите от лекция 2

Броят на необходимите букви за изписването на
всички числа между 1 и 1000 (включително) на английски

```java
public class Exercise4 {
    private static final String ONES[] = {"", "one", "two", "three", "four", "five", "six", "seven", "eight", "nine"};
    private static final String TEENS[] = {"", "eleven", "twelve", "thirteen", "fourteen", "fifteen", "sixteen", "seventeen", "eighteen", "nineteen"};
    private static final String TENS[] = {"", "ten", "twenty", "thirty", "forty", "fifty", "sixty", "seventy", "eighty", "ninety"};
    private static final String HUNDRED = "hundred";
    private static final String AND = "and";
    private static final String ONE_THOUSAND = "onethousand";

    public static void main(String[] args) {
        StringBuilder temp = new StringBuilder();
        int length = 0;

        for (int i = 1; i <= 1000; i++) {
            if (i == 1000) {
                temp.append(ONE_THOUSAND);
            } else {
                int hundreds = i / 100;
                int hundredRemainder = i % 100;

                if (hundreds > 0) {
                    temp.append(ONES[hundreds]).append(HUNDRED);
                    if (hundredRemainder != 0) {
                        temp.append(AND);
                    }
                }

                if (hundredRemainder > 10 && hundredRemainder < 20) {
                    temp.append(TEENS[(hundredRemainder) % 10]);
                } else if (hundredRemainder < 10) {
                    temp.append(ONES[hundredRemainder]);
                } else {
                    temp.append(TENS[(hundredRemainder) / 10]);
                    if ((hundredRemainder) % 10 > 0) {
                        temp.append(ONES[(hundredRemainder) % 10]);
                    }
                }
            }

            length += temp.length();
            temp.replace(0, temp.length(), "");
        }

        System.out.printf("The number of letters is: %d %n", length);
    }
}
```

Обектно-ориентирано програмиране

Класове и обекти. Наследяване и полиморфизъм.
Интерфейси и вътрешни класове

Обектно-ориентирано програмиране (ООП)

- Какво е ООП?
- Основни понятия
  - клас - шаблон
  - обект - инстанция на шаблон
    - състояние (state)
    - поведение (behaviour)

Пример: Студент

Клас Студент

- Атрибути
  - Име
  - Факултетен номер
  - Среден успех
- Поведение
  - Ходи на лекции
  - Ходи на изпити
  - Преписва

Обект Пешо от клас Студент

- Състояние
  - Пешо
  - 1234567890
  - 3.25
- Поведение
  - Отива на лекция по ООП
  - Отива на изпит по математика
  - Не преписва

Основни понятия при класовете

- Атрибути
- Поведение
- Енкапсулация
- Класово състояние/поведение
- Наследяване

Основни понятия при обектите

- Състояние
- Поведение
- Идентичност

Отношения между класовете

- Зависимост (coupling, dependency, uses-a)
- Агрегация (has-a)
- Наследяване (is-a)

Създаване на обекти

- Конструктор
- ключовата дума: new

```
Date currentDate = new Date();
```

Променливи от референтен тип

- НЕ са обекти
- НЕ съдържат обекти
- Съдържат само препратка (референция, адрес) на обект
- Няколко променливи могат да сочат към един обект
- Променливите НЕ са инициализирани с null по подразбиране

Класове от стандартната библиотека

- java.util.Date - представя дата
- java.util.Random - генератор на случайни числа
- java.util.Scanner - вход от поток
- java.lang.System - агрегатор на системни операции
- java.util.GregorianCalendar - представя дата по грегорианския (нашия) календар

Създаване на клас Студент

- Полета
- Конструктори
- Методи

```java
public class Student {
    private String name;
    private String facultyNumber;
    private double averageResults;

    public Student() { … }
    public Student(String name, String facultyNumber) { … }
    public String getFacultyNumber() {
        return this.facultyNumber;
    }
    public void setFacultyNumber(String newFacultyNumber) {
        this.facultyNumber = newFacultyNumber;
    }
    public void attendLecture() { ... }
    public void goToExam() { ... }
    public void cheat() { ... }
}
```

Изследване на клас Студент

- Модификатори за достъп
- Енкапсулация

Предимства на енкапсулацията

- Валидиране на данни
- Защитаване на данни от промени
- Гъвкавост - възможно е да бъде променена вътрешна имплементация без промени на публичното API

Модификатора final

- Приложен към примитивен тип
- Приложен към обект
- Приложен към клас
- Приложен към метод

Статични(класови) полета и методи

- Асоциирани са съм самия клас, а не обектите инстанцирани от него
- Достъпни са както посредством класа, така и посредство обектите от него
- Статични полета често се използват за съхранение на константи
- Статични методи често се използват за реализация на фабрични методи

Параметри на методите

- Предаване по стойност
- Предаване на препратка към обект
- Прототип на метод (сигнатура)
  - име
  - параметри

Пример 1

```
public static void main(String[] args) {
    int passedValue = 3;
    System.out.println(passedValue);   // 3
    printModifiedValue(passedValue);
    System.out.println(passedValue);   // 3
}

private static void printModifiedValue(int value) {
    value += 10;
    System.out.println(value); // 13
}
```

Пример 2

```
public class ParameterExamples2 {
    public static void main(String[] args) {
        Employee theNewTeacher = new Employee("Pesho Peshev", 1000);
        System.out.printf("Salary: %d %n", theNewTeacher.getSalary()); // 1000
        changeSalary(theNewTeacher, 1200);
        System.out.printf("Salary: %d %n", theNewTeacher.getSalary()); // 1200
    }

    private static void changeSalary(Employee employee, int amount) {
        employee.setSalary(amount);
        System.out.printf("Changed Salary: %d %n", employee.getSalary()); // 1200
    }
}
```

Конструиране на обекти

- Посредством конструктор
- Конструктор по подразбиране
- Всички полета се инициализират със стойност по подразбиране
- Може да имате повече от един конструктор
- Може да извиквате един конструктор от друг

```
public class Student {
    private String facultyNumber;
    private String name;

    public Student(String name) {
        this.name = name;
    }
    public Student(String name, String fNumber) {
        this(name);
        this.facultyNumber = fNumber;
    }
}
```

```
public static void main(String[] args) {
    Student st = new Student("Pesho", "1234567890");
    System.out.println(st.getName());
}
```

Презареждане (overloading) на методи

- Няколко метода с едно и също име
- Сигнатура

Пример: Презареждане

```
public void drive() {
    fun += 1;
}
public void drive(boolean fast) {
    if (fast)
        fun += 3;
}
public void drive(boolean fast, String car) {
    if (fast && car.equals("Lamborghini"))
        fun += 20;
}
```

Пакети в Java

- групиране на свързани класове
- предотвратяват конфликти в имената на класове
- съответстват на директории във файловата система
- използване на домейн за уникалност
- запазени имена на пакети - java.*, javax.*
- ключовата дума package

Импортиране на класове и пакети

- Ключовата дума import
- Класове от същия пакет са достъпни без импорт
- Достъп до произволен клас чрез пълен път (package + class)
- wildcard import, не се препоръчва

Статични импорти

- Импортиране само на определени статични елементи от даден клас - полета или методи
- Подходяща техника за работа със статични класове като java.util.Math
- Могат да нарушат четимостта на програмата, ако се използват безразборно

Пакетиране на Java код

- jar архив
  - bytecode + мета информация, компресирани със zip
- Изпълнение:
  - java -jar somejar.jar MainClass

Java Classpath

- Указва местата, където виртуалната машина търси класове, при изпълнението на дадена програма
- може да се задава като параметър от командния ред или като настройка на средата
- Източник на много проблеми

Classpath пример

- В UNIX(Linux, Solaris, BSD)
- java -cp .:dir2:dir3 MyClass
- export CLASSPATH=.:dir2:dir3
- В Windows:
- Java -cp .;dir2;dir3; MyClass
- set CLASSPATH=.;dir2;dir3

Документиране на класове с javadoc

- Разглеждане на javadoc документация
- онлайн
- в интегрираната среда за разработка
- Документира се само публичния интерфейс на едно API
- Стандартни javadoc анотации
- Работа с инструмента javadoc
    - javadoc -d docdir package

Наследяване

- Фундаментална техника в ООП
- Изграждане на нови класове върху основата на съществуващи класове
- Преизползват се методите и полетата на съществуващи класове
- Добавя се ново поведение и състояние
- Променя се съществуващото поведение

Концептуален пример

```
class Convertible {
//  Key (private)
//   Speed : 155 (miles / hour)
//    Weight 1600 kg
//   Engine : 3.2 L S54 inline-6
}
```

```
class Roadster extends Convertible {

 //   Speed : 165 (miles / hour)
//    Weight 1399 kg
}
```

От общото към частното

- наследяване == разширяване
- суперклас(клас родител/базов клас)
- подклас(клас наследник)
- отношението между подклас и суперклас е е(is-a)
  - Планинското колело е колело.
  - Програмистът е човек.

Модификатори за достъп

- Ограничават видимостта на класовете и техните членове
- private
  - достъп само в рамките на класа
- public
  - достъп за всички
- default(package)
  - достъп за всички в пакета
- protected
  - default + достъп в наследените

Ключови моменти при наследяването

- Наследява на public и protected членове
- Наследява default, ако е в същия пакет
- private елементите не са достъпни
- Конструкторът на подкласа извиква конструктора на суперкласа
- Ключовата дума super

Какво можете да правите в подклас?

- Директен достъп до наследени методи и полета
- Предефиниране на методи и полета
- НЕ предефинирайте полета
- Дефиниране на нови методи и полета
- Извикване конструктор от суперкласа

Йерархия на наследяването

- Единична йерархия на наследяване - един подклас може да има само един суперклас
- Избягват се много от проблемите на C++
- Интерфейсите предлагат подобие на множествено наследяване

Полиморфизъм

- Променливи от суперклас могат да бъдат асоциирани с обекти от подкласове
- Виртуалната машина знае истинските типове на обектите и по време на изпълнение извиква техните методи, независимо с променлива от какъв тип са асоциирани
- Полиморфизмът е тясно свързан с късното свързване на методите(late binding).

Пример

```
Employee[] staff = new Employee[3];

staff[0] = new Manager("Big Boss", 5000, 1000);
staff[1] = new Employee("Ivan Ivanov", 2000);
staff[2] = new Employee("Pesho Peshev", 3000);


for (Employee e: staff) {
    System.out.printf("$%4.0f", e.getSalary());
}
```

Предимства на полиморфизма

- Ниска свързаност(loose coupling)
- Гъвкавост
- Простота

Късно свързване

- Виртуалната машина знае истинския тип на всеки един обект
- Виртуалната машина изпълнява методите на базата на истинския тип
- Необходимост за реализиране на полиморфизъм

Забраняване на наследяването и късното свързване

- final class - неразширяем клас
- final method - метод, който не може да бъде динамично свързан
- Във final class всички методи са имплицитно final
- final като оптимизационна и защитна техника

Абстрактни класове

- Създадени да бъдат разширявани
- Не могат да бъдат инстанцирани
- Обикновено са на върха на йерархията на наследяването
- Съдържат един или повече методи без имплементация(абстрактни методи)

Пример 1/3

```
public abstract class Person {
    private String name;
    public Person(String name) {
        this.name = name;
    }
    public String getName() {
        return this.name;
    }
    public abstract String getDescription();
}
```

Пример 2/3

```
public class Student extends Person {
    private String facultyNumber;
    public Student(String name) {
        super(name);
    }
    public String getDescription() {
        return getName() + " is a student");
    }
}
```

Пример 3/3

```
public class Employee extends Person {
    private double salary;
    public Employee(String name, double salary) {
        super(name);
        this.salary = salary;
    }
    public String getDescription() {
        return getName() + " has salary $"+ this.salary;
    }
}
```

Преобразуване на обекти

- Всеки обект от подклас и е обект от суперкласа
- Обратното не е вярно
- Референция от суперклас може да бъде конвертирана до референция от подклас изрично
- Изричното преобразуване е опасна операция, която може да доведе го грешки по време на изпълнение на програмата

Преобразуване на обекти

- Сигурно преобразуване с instanceof

```
if(e instanceof Programmer) {
        p = (Programmer) e;
}
```

- Невъзможните преобразувания се улавят от компилатора
- Date d = (Date) e ;

Върховният суперклас Object

- Начало на класовата йерархия в Java
- Всеки клас имплицитно наследява Object
- Съдържа в себе си методи, които имат смисъл за всички обекти
  - clone()
  - equals()
  - hashCode()
  - toString()
- Всяка референция може да бъде преобразувана до тип Object

Сравняване на обекти

- == сравнява референции - сравнение за идентичност
- метода equals()
- Дефиниране на подходящ метод equals()
  - проверка за сравнение с null
  - проверка за сравнение със същия обект
  - проверка за сравнение с различен тип
  - сравняване на двата обекта по полета
  - рекурсивно сравняване на полета от референтни типове с equals
- връзка между equals и hashCode

Модификатор за достъп protected

- Еквивалент на default достъп + директен достъп в подкласовете
- Употребата му се счита за лош стил
- четири нива на достъп
  - private
  - public
  - default(package)
  - protected

Интерфейс

- Наборът от методи, които един клас предлага
- Езикова конструкция, която съдържа контракт(набор от задължения)
- Класовете могат да изпълнят контракта(да имплементират интерфейса)
- Обикновено съдържа само методи
- Методите са с ниво на достъп public и са абстрактни

Дефиниция на интерфейс

```
interface SomeInterface {

    Type field1; // bad style

    Type field2;

    ...

    Type method1();

    Type method2();

}
```

Имплементиране на интерфейс

```
class SomeClass implements SomeInterface {
    ...
    @Override
    public Type method1() {...}
    public Type method3() {...}
}
```

Особености на интерфейсите

- Всички методи в тях са абстрактни
- Един клас може да имплементира повече от един интерфейс
- Интерфейсът НЕ Е клас - не може създавате обекти от интерфейс
- Могат да бъдат декларирани променливи от интерфейсен тип
- Работят с instanceof оператора
- Не е желателно да имат полета
- Абстрактен клас може да имплементира интерфейс напълно или частично

Особености на интерфейсите

- Не могат да съдържат instance полета
- Не могат да съдържа статични методи
- Всички полета в един интерфейс са на практика константи(public static final)
- Един интерфейс може да разшири(наследи) друг
- Интерфейси без методи се използват като маркери(tags)

Сравняване на обекти с интерфейса Comparable

```
public interface Comparable<T> {
    int compareTo(T other);
}
```

- Използва се за сравняване на обекти с естествена подредба
- Генеричен интерфейс от Java 5

Comparable - пример

```
class Employee implements Comparable<Employee> {
    private double salary;
    ...
    public int compareTo(Employee other) {
        if ( salary < other.salary ) return -1;
        if ( salary > other.salary ) return 1;
        return 0;
    }
}
```

Вътрешни класове

- Влагане да дефиницията на един клас в друг
- Реализирани са на ниво компилатор
- Позволяват достъп до членовете на класа, в който са вложени
- Видове
  - стандартни
  - локални
  - анонимни
  - статични

Стандартни вътрешни класове

- Дефинирани са в друг клас на нивото на полетата и методите му
- Имат достъп до полетата и методите на външния клас
- Имат скрита референция към външния клас
- Всяка инстанция от външния клас носи дефиницията на вътрешния

Локални вътрешни класове

- Дефинирани са в тялото на метод на външния клас
- Имат достъп до локалните променливи в метода но само до тези маркирани като nal
- Не са видими извън метода, в който са дефинирани

Анонимни вътрешни класове

- Разновидност на локалните класове
- Създава се обект от клас дефиниран след оператора за присвояване
- Обикновено анонимните класове имплементират някой интерфейс
- Не могат да дефинират собствени конструктори

Статични вътрешни класове

- Еквивалентни на стандартните, но без референция към външния клас
- Служат като допълнително пространство на имената

Упражнение 1

- Реализирайте клас Rational, който представя дроби и предоставя математически операции за работа с тях - събиране, изваждане, умножение и делене. Всички негови методи трябва да могат да работят освен с други дроби и с цели числа.

Упражнение 2

- Моделирайте йерархия от геометрични обекти:
  - Circle
  - Rectangle
  - Square
- За всички от тях трябва да имаме възможност да пресмятаме обиколката и площта им. Използвайте абстрактен базов клас като корен на тази йерархия от класове.
- Създайте 10 произволни фигури и ги запишете в масив
- За всеки фигура в масива отпечатайте нейните измерения, периферия и площ.