Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

**факультет программной инженерии и компьютерной техники**

**ЛАБОРАТОРНАЯ РАБОТА № 3**

по дисциплине

‘Программирование’

Вариант №5218

*Выполнил:*

Студент группы P3109

Суханкин Дмитрий Юрьевич

*Преподаватель:*

Шешуков Дмитрий

Михайлович



Санкт-Петербург, 2021

Оглавление

[Цель 2](#_Toc85140136)

[Задание 3](#_Toc85140137)

[Диаграмма классов реализованной объектной модели 5](#_Toc85140138)

[Исходный код программы 5](#_Toc85140139)

[Lab1.java 5](#_Toc85140140)

[Pokemon 6](#_Toc85140141)

[Sawk.java 6](#_Toc85140142)

[Binacle.java 6](#_Toc85140143)

[Barbaracle.java 7](#_Toc85140144)

[Cleffa.java 7](#_Toc85140145)

[Clefairy.java 8](#_Toc85140146)

[Clefable.java 8](#_Toc85140147)

[Moves 8](#_Toc85140148)

[BulkUp.java 8](#_Toc85140149)

[FireBlast.java 9](#_Toc85140150)

[Headbutt.java 10](#_Toc85140151)

[Payback.java 10](#_Toc85140152)

[Pound.java 11](#_Toc85140153)

[Quash.java 11](#_Toc85140154)

[RockTomb.java 11](#_Toc85140155)

[ShiftGear.java 12](#_Toc85140156)

[SpacialRend.java 12](#_Toc85140157)

[Swagger.java 13](#_Toc85140158)

[Результат работы программы 13](#_Toc85140159)

[Выводы 16](#_Toc85140160)

# Задание

1. Доработанная модель должна соответствовать принципам SOLID.
2. Программа должна содержать как минимум два интерфейса и один абстрактный класс (номенклатура должна быть согласована с преподавателем).
3. В разработанных классах должны быть переопределены методы equals(), toString() и hashCode().
4. Программа должна содержать как минимум один перечисляемый тип (enum).

# Текст

# Особенно среди всего этого великолепия выделялось одно огромнейшее чертово колесо, которое мало того что вертелось, как обычное чертово колесо, но еще в то же время вихлялось в разные стороны, словно собиралось свалиться на головы прохожим. Тысячи коротышек карабкались вверх по лестницам, чтобы покачаться на качелях, потрястись на заводных деревянных лошадках, прокатиться над улицей по канату на специальном велосипеде, покружиться на карусели, или хотя бы на чертовом колесе.

# Диаграмма классов реализованной объектной модели

implements

extends

extends

extends

extends

extends

Ride

*Abstract*

Interactable

*Interface*

Horse

MerryGoRound

Bicycle

Swing

HeckingWheel

Result

*Enum*

Shorty

implements

Interactor

*Interface*

В этот раз даже немного красиво вышло.

# Исходный код программы

Можно почитать на GitHub: <https://github.com/suhankins/ITMO-Labs/tree/main/Programming/lab2/src>

## Hell

### Main.java

package hell;

import hell.rides.\*;

import hell.states.\*;

public class Main {

    public static void main(String[] args) {

        Ride[] rides = {new HeckingWheel(), new Swing(), new Horse(), new Bicycle(), new MerryGoRound()};

        //In the description it's said that there were "thousands" of shorties,

        //but i don't it would be a good idea to create a thousand of objects

        Shorty[] shorties = new Shorty[100];

        for (int i = 0; i < shorties.length; i++) {

            shorties[i] = new Shorty();

            shorties[i].interact(rides[(int)Math.floor((Math.random() \* rides.length))]);

        }

    }

}

### Interactor.java

package hell;

import hell.rides.\*;

//I only added this beause I need 2 interfaces and I only had 1

public interface Interactor {

    public void interact(Ride ride);

}

### Shorty.java

package hell;

import hell.rides.\*;

import hell.states.\*;

public class Shorty implements Interactor {

    public int id;

    private static int shortiesCount = 0;

    public Shorty() {

        shortiesCount++;

        id = shortiesCount;

    }

    public void interact(Ride ride) {

        if (!ride.canBeInteractedWith()) {

            if (Math.random() > 0.1) {

                System.out.printf("Shorty %d coudldn't ride %s!%n", this.id, ride.getName());

            } else {

                ride.setCondition(true);

                this.interact(ride);

            }

        } else {

            String flavorText = "";

            switch(ride.interactionResult()){

                case HADFUN:

                    flavorText = "had fun on";

                    break;

                case FELLOFF:

                    flavorText = "fell of";

                    break;

                case WASBORED:

                    flavorText = "was bored on";

                    break;

                default:

                    flavorText = "had something unspeakable happen to them on";

                    break;

            }

            System.out.printf("%s %s the %s%n", this.toString(), flavorText, ride.getName());

        }

    }

    @Override

    public boolean equals(Object o){

        if (o == this) {

            return true;

        }

        if (!(o instanceof Shorty)) {

            return false;

        }

        Shorty s = (Shorty)o;

        return s.id == this.id;

    }

    @Override

    public String toString() {

        return String.format("Shorty %d", this.id);

    }

    @Override

    public int hashCode() {

        final int prime = 31;

        int result = 1;

        result = prime \* result + id;

        return result;

    }

}

## Hell.states

### Result.java

package hell.states;

public enum Result {

    HADFUN,

    FELLOFF,

    WASBORED,

    COULDNT;

}

## Hell.rides

### Interactable.java

package hell.rides;

import hell.states.\*;

public interface Interactable {

    public boolean canBeInteractedWith();

    public Result interactionResult();

}

### Ride.java

package hell.rides;

import hell.states.\*;

public abstract class Ride implements Interactable {

    private boolean condition = true;

    private String name;

    //Name stuff

    public Ride(String name) {

        this.name = name;

    }

    public String getName() {

        return name;

    }

    //Condition stuff

    //This one can be overriden, that's why we need getCondition

    public boolean canBeInteractedWith() {

        return condition;

    }

    public boolean getCondition() {

        return condition;

    }

    public void setCondition(boolean bool) {

        condition = bool;

        if (!condition) System.out.printf("%s is broken! ", this.getName());

        else System.out.printf("%s was fixed! ", this.getName());

    }

    @Override

    public boolean equals(Object o){

        if (o == this) {

            return true;

        }

        if (!(o instanceof Ride)) {

            return false;

        }

        Ride s = (Ride)o;

        return s.getName() == this.name

               && s.getCondition() == this.condition;

    }

    @Override

    public String toString() {

        return String.format("Ride %s; is broken: %b", this.getName(), !this.condition);

    }

    @Override

    public int hashCode() {

        final int prime = 31;

        int result = 1;

        result = prime \* result + getName().hashCode() + (getCondition() ? 1 : 0);

        return result;

    }

}

### Bicycle.java

package hell.rides;

import hell.states.\*;

public class Bicycle extends Ride {

    //There are only 2 bicycles and if both break -

    //ride can no longer be interacted with

    public int breakCount = 0;

    public Bicycle() {

        super("Bicycle");

    }

    @Override

    public boolean canBeInteractedWith() {

        //If both bicycles are broken ride can no

        //longer be interacted with

        return breakCount < 2;

    }

    @Override

    public void setCondition(boolean bool) {

        super.setCondition(bool);

        //resetting break counter

        if (getCondition()) breakCount = 0;

    }

    public Result interactionResult() {

        double diceRoll = Math.random();

        //75% user had fun

        //10% user was bored

        //15% user fell off

        if (diceRoll > 0.25) {

            return Result.HADFUN;

        } else if (diceRoll > 0.15) {

            return Result.WASBORED;

        } else {

            setCondition(false);

            breakCount++;

            return Result.FELLOFF;

        }

    }

    //overriding equals because we need to take into account

    //break count too

    @Override

    public boolean equals(Object o){

        if (o == this) {

            return true;

        }

        if (!(o instanceof Bicycle)) {

            return false;

        }

        Bicycle s = (Bicycle)o;

        return s.getName() == this.getName()

               && s.getCondition() == this.getCondition()

               && s.breakCount == this.breakCount;

    }

    //overriding tostring to print break count

    @Override

    public String toString() {

        return String.format("Ride %s; is broken: %b; broken bicycles: %d", this.getName(), !this.getCondition(), this.breakCount);

    }

    @Override

    public int hashCode() {

        final int prime = 31;

        int result = 1;

        result = prime \* result + getName().hashCode() + (getCondition() ? 1 : 0) + breakCount;

        return result;

    }

}

### HeckingWheel.java

package hell.rides;

import hell.states.\*;

public class HeckingWheel extends Ride {

    public HeckingWheel() {

        super("Hecking Wheel");

    }

    @Override

    public boolean canBeInteractedWith() {

        //No one cares if hecking wheel is broken because it \*being broken\* is the point

        return true;

    }

    public Result interactionResult() {

        if (Math.random() < 0.1 && getCondition()) {

            setCondition(false);

        }

        double diceRoll = Math.random();

        if (!getCondition()) {

            diceRoll /= 2;

        }

        //60% had fun

        //15% was bored

        //25% fell off

        //After breaking

        //20% had fun

        //30% was bored

        //50% fell off

        if (diceRoll > 0.4) {

            return Result.HADFUN;

        } else if (diceRoll > 0.25) {

            return Result.WASBORED;

        } else {

            return Result.FELLOFF;

        }

    }

}

### Horse.java

package hell.rides;

import hell.states.\*;

public class Horse extends Ride {

    public Horse() {

        super("Wooden Horse");

    }

    public Result interactionResult() {

        double diceRoll = Math.random();

        //75% user had fun

        //15% user was bored

        //10% user fell off

        if (diceRoll > 0.25) {

            return Result.HADFUN;

        } else if (diceRoll > 0.1) {

            return Result.WASBORED;

        } else {

            setCondition(false);

            return Result.FELLOFF;

        }

    }

}

### MerryGoRound.java

package hell.rides;

import hell.states.\*;

public class MerryGoRound extends Ride {

    public MerryGoRound() {

        super("Merry-Go-Round");

    }

    public Result interactionResult() {

        double diceRoll = Math.random();

        //35% user had fun

        //64% user was bored

        //1% user fell off

        if (diceRoll > 0.65) {

            return Result.HADFUN;

        } else if (diceRoll > 0.01) {

            return Result.WASBORED;

        } else {

            setCondition(false);

            return Result.FELLOFF;

        }

    }

}

### Swing.java

package hell.rides;

import hell.states.\*;

public class Swing extends Ride {

    public Swing() {

        super("Swing");

    }

    public Result interactionResult() {

        double diceRoll = Math.random();

        //65% user had fun

        //25% user was bored

        //10% user fell off

        if (diceRoll > 0.35) {

            return Result.HADFUN;

        } else if (diceRoll > 0.1) {

            return Result.WASBORED;

        } else {

            setCondition(false);

            return Result.FELLOFF;

        }

    }

}

# Результат работы программы

Shorty 1 was bored on the Hecking Wheel

Shorty 2 was bored on the Merry-Go-Round

Shorty 3 had fun on the Swing

Shorty 4 was bored on the Merry-Go-Round

Shorty 5 had fun on the Wooden Horse

Shorty 6 had fun on the Bicycle

Shorty 7 had fun on the Swing

Shorty 8 was bored on the Bicycle

Shorty 9 was bored on the Hecking Wheel

Shorty 10 had fun on the Bicycle

Shorty 11 had fun on the Hecking Wheel

Shorty 12 had fun on the Swing

Shorty 13 was bored on the Merry-Go-Round

Shorty 14 had fun on the Hecking Wheel

Shorty 15 had fun on the Wooden Horse

Shorty 16 had fun on the Merry-Go-Round

Shorty 17 had fun on the Bicycle

Shorty 18 was bored on the Swing

Shorty 19 had fun on the Swing

Shorty 20 was bored on the Merry-Go-Round

Shorty 21 had fun on the Swing

Shorty 22 had fun on the Merry-Go-Round

Shorty 23 was bored on the Swing

Shorty 24 had fun on the Wooden Horse

Shorty 25 was bored on the Merry-Go-Round

Shorty 26 had fun on the Wooden Horse

Shorty 27 was bored on the Merry-Go-Round

Shorty 28 had fun on the Bicycle

Shorty 29 had fun on the Bicycle

Shorty 30 had fun on the Bicycle

Shorty 31 was bored on the Merry-Go-Round

Bicycle is broken! Shorty 32 fell of the Bicycle

Shorty 33 was bored on the Wooden Horse

Shorty 34 had fun on the Swing

Shorty 35 fell of the Hecking Wheel

Shorty 36 had fun on the Wooden Horse

Shorty 37 was bored on the Merry-Go-Round

Shorty 38 had fun on the Wooden Horse

Shorty 39 had fun on the Swing

Shorty 40 was bored on the Swing

Swing is broken! Shorty 41 fell of the Swing

Shorty 42 was bored on the Merry-Go-Round

Shorty 43 had fun on the Wooden Horse

Bicycle is broken! Shorty 44 fell of the Bicycle

Shorty 45 was bored on the Hecking Wheel

Shorty 46 had fun on the Hecking Wheel

Shorty 47 fell of the Hecking Wheel

Bicycle was fixed! Bicycle is broken! Shorty 48 fell of the Bicycle

Моя любимая часть. Коротышка 48 починил велосипеды и тут же один сломал.

Shorty 49 was bored on the Wooden Horse

Shorty 50 had fun on the Wooden Horse

Shorty 51 had fun on the Merry-Go-Round

Shorty 52 had fun on the Hecking Wheel

Bicycle is broken! Shorty 53 fell of the Bicycle

Shorty 54 was bored on the Merry-Go-Round

Shorty 55 coudldn't ride Bicycle!

Shorty 56 coudldn't ride Bicycle!

Shorty 57 had fun on the Wooden Horse

Shorty 58 coudldn't ride Swing!

Shorty 59 coudldn't ride Bicycle!

Shorty 60 coudldn't ride Bicycle!

Shorty 61 had fun on the Hecking Wheel

Shorty 62 coudldn't ride Swing!

Shorty 63 coudldn't ride Swing!

Wooden Horse is broken! Shorty 64 fell of the Wooden Horse

Shorty 65 coudldn't ride Swing!

Shorty 66 was bored on the Merry-Go-Round

Shorty 67 had fun on the Hecking Wheel

Shorty 68 was bored on the Merry-Go-Round

Shorty 69 was bored on the Merry-Go-Round

Shorty 70 coudldn't ride Bicycle!

Shorty 71 had fun on the Hecking Wheel

Shorty 72 was bored on the Hecking Wheel

Swing was fixed! Shorty 73 had fun on the Swing

Shorty 74 was bored on the Merry-Go-Round

Shorty 75 had fun on the Merry-Go-Round

Shorty 76 was bored on the Merry-Go-Round

Shorty 77 had fun on the Swing

Shorty 78 had fun on the Swing

Shorty 79 had fun on the Hecking Wheel

Shorty 80 coudldn't ride Wooden Horse!

Shorty 81 coudldn't ride Wooden Horse!

Shorty 82 fell of the Hecking Wheel

Shorty 83 was bored on the Merry-Go-Round

Merry-Go-Round is broken! Shorty 84 fell of the Merry-Go-Round

Shorty 85 coudldn't ride Bicycle!

Merry-Go-Round was fixed! Shorty 86 was bored on the Merry-Go-Round

Shorty 87 was bored on the Merry-Go-Round

Shorty 88 was bored on the Hecking Wheel

Shorty 89 coudldn't ride Wooden Horse!

Shorty 90 was bored on the Merry-Go-Round

Shorty 91 had fun on the Hecking Wheel

Shorty 92 had fun on the Swing

Hecking Wheel is broken! Shorty 93 had fun on the Hecking Wheel

Shorty 94 had fun on the Swing

Wooden Horse was fixed! Shorty 95 had fun on the Wooden Horse

Shorty 96 was bored on the Hecking Wheel

Shorty 97 was bored on the Merry-Go-Round

Shorty 98 was bored on the Swing

Shorty 99 had fun on the Swing

Shorty 100 fell of the Hecking Wheel

# Выводы

Это гораздо более интересный пример использования ООП чем наследование классов покемонов, которые вообще и наследовать смысла нет ведь покемоны отличаются только статами.