Zjazd 1

# Variables

public class Variables {  
  
 public static void main(String[] args) {  
 System.*out*.println("Hello, Pjatk");  
 int myFirstNumber = 35;  
 System.*out*.println(myFirstNumber);  
 }  
  
}

# Expressions

public class Expressions {  
 public static void main(String[] args) {  
 System.*out*.println("Hello, Pjatk");  
 int myFirstNumber = 35;  
 int mySecondNumber = 12;  
 int myThirdNumber = myFirstNumber \* 2;  
 int myTotal = myFirstNumber + mySecondNumber + myThirdNumber;  
 System.*out*.println(myTotal);  
 int myLastOne = 1000 - myTotal;  
 System.*out*.println(myLastOne);  
 }  
}

# Numeric types

public class IntByteShortLong {  
 public static void main(String[] args) {  
  
 int myValue = 10000;  
  
 int myMinIntValue = Integer.*MIN\_VALUE*;  
 int myMaxIntValue = Integer.*MAX\_VALUE*;  
 System.*out*.println("Integer Minimum Value = " + myMinIntValue);  
 System.*out*.println("Integer Maximum Value = " + myMaxIntValue);  
 System.*out*.println("Busted MAX value = " + (myMaxIntValue + 1));  
 System.*out*.println("Busted MIN value = " + (myMinIntValue - 1));  
  
 int myMaxIntTest = 2\_147\_483\_647;  
  
 byte myMinByteValue = Byte.*MIN\_VALUE*;  
 byte myMaxByteValue = Byte.*MAX\_VALUE*;  
 System.*out*.println("Byte Minimum Value = " + myMinByteValue);  
 System.*out*.println("Byte Maximum Value = " + myMaxByteValue);  
  
 short myMinShortValue = Short.*MIN\_VALUE*;  
 short myMaxShortValue = Short.*MAX\_VALUE*;  
 System.*out*.println("Short Minimum Value = " + myMinShortValue);  
 System.*out*.println("Short Maximum Value = " + myMaxShortValue);  
  
 long myLongValue = 100;  
 long myMinLongValue = Long.*MIN\_VALUE*;  
 long myMaxLongValue = Long.*MAX\_VALUE*;  
 System.*out*.println("Long Minimum Value = " + myMinLongValue);  
 System.*out*.println("Long Maximum Value = " + myMaxLongValue);  
 long bigLongLiteralValue = 2\_147\_483\_647\_234L;  
 System.*out*.println(bigLongLiteralValue);  
  
 short bigShortLiteralValue = 32767;  
 }  
}

# Cast

int myTotal = (myMinIntValue / 2);  
  
byte myNewByteValue = (byte) (myMinByteValue / 2);  
  
short myNewShortValue = (short) (myMinShortValue / 2);

# Float&double

public class FloatAndDouble {  
 public static void main(String[] args) {  
  
 float myMinFloatValue = Float.*MIN\_VALUE*;  
 float myMaxFloatValue = Float.*MAX\_VALUE*;  
 System.*out*.println("Float minimum value = " + myMinFloatValue);  
 System.*out*.println("Float maximum value = " +myMaxFloatValue);  
  
 double myMinDoubleValue = Double.*MIN\_VALUE*;  
 double myMaxDoubleValue = Double.*MAX\_VALUE*;  
 System.*out*.println("Double minimum value = " + myMinDoubleValue);  
 System.*out*.println("Double maximum value = " +myMaxDoubleValue);  
  
 int myIntValue = 5 / 3;  
 float myFloatValue = 5f / 3f;  
 double myDoubleValue = 5.00 / 3.00;  
 System.*out*.println("MyIntValue= " + myIntValue);  
 System.*out*.println("MyFloatValue= " + myFloatValue);  
 System.*out*.println("MyDoubleValue= " + myDoubleValue);  
  
 }  
}

# Strings

public class Strings {  
 public static void main(String[] args) {  
 String myString = "This is a string";  
 System.*out*.println("myString is equal to " + myString);  
 myString = myString + ", and this is more.";  
 System.*out*.println("myString is equal to " + myString);  
 myString = myString + " \u00A9 2019";  
 System.*out*.println("myString is equal to " + myString);  
 String numberString = "250.55";  
 numberString = numberString + "49.95";  
 System.*out*.println(numberString);  
 String lastString = "10";  
 int myInt = 50;  
 lastString = lastString + myInt;  
 System.*out*.println("LastString is equal to " + lastString);  
 double doubleNumber = 120.47d;  
 lastString = lastString + doubleNumber;  
 System.*out*.println("LastString is equal to " + lastString);  
  
  
 }  
}

# If else

public class IfElseStatement {  
  
 public static void main(String[] args) {  
  
 boolean isAlien = false;  
 if (isAlien == false) {  
 System.*out*.println("It is not an alien!");  
 System.*out*.println("And I am scared of aliens");  
 }  
  
 int topScore = 80;  
 if (topScore < 100) {  
 System.*out*.println("You got the high score!");  
 }  
  
 int secondTopScore = 95;  
 if ((topScore > secondTopScore) && (topScore < 100)) {  
 System.*out*.println("Greater than second top score and less than 100");  
 }  
  
 if ((topScore > 90) || (secondTopScore <= 90)) {  
 System.*out*.println("Either or both of the conditions are true");  
 }  
  
 int newValue = 50;  
 if (newValue == 50) {  
 System.*out*.println("This is true");  
 }  
  
 boolean isCar = false;  
 if (!isCar) {  
 System.*out*.println("This is not supposed to happen");  
 }  
 }  
}

Zjazd 2

# Classes

Utwórz klasę o nazwie Kwadrat. Zadeklaruj tam dwa pola prywatne "x" oraz "y", dodaj gettery, settery i konstruktory. Utwórz metodę liczącą pole

public class Rectangle {  
 private int x;  
 private int y;  
  
 public int calculateArea() {  
 return this.x \* this.y;  
 }  
  
 public Rectangle(int x, int y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public int getX() {  
 return x;  
 }  
  
 public void setX(int x) {  
 this.x = x;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public void setY(int y) {  
 this.y = y;  
 }  
}

# Method overloading

Utwórz drugą metodę o takiej samej nazwie jaką ma calculateArea, tylko zamiast liczyc pole dla argumentów klasy Rectangle, niech przyjmie argumenty z zewnątrz.

public class Rectangle {  
 private int x;  
 private int y;  
  
 public int calculateArea() {  
 return this.x \* this.y;  
 }  
  
 public int calculateArea(int x, int y) {  
 return x \* y;  
 }  
  
 public Rectangle(int x, int y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 public int getX() {  
 return x;  
 }  
  
 public void setX(int x) {  
 this.x = x;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public void setY(int y) {  
 this.y = y;  
 }  
}

# Inheritance

Utworzyc klase Car z polami (numberOfDoors, gears) klase Tesla w która dziedziczy po samochodzie i dodatkowo posiada ładowarke i metode "ładuj".Utworzyć trzecia klase "ModelS" ktora dziedziczy po Tesli i posiada 700 koni i rowniez ma swoje metody

public class Car {  
 private int numberOfDoors;  
 private int gears;  
  
 public Car(int numberOfDoors, int gears) {  
 this.numberOfDoors = numberOfDoors;  
 this.gears = gears;  
 }  
  
 public int getNumberOfDoors() {  
 return numberOfDoors;  
 }  
  
 public void setNumberOfDoors(int numberOfDoors) {  
 this.numberOfDoors = numberOfDoors;  
 }  
  
 public int getGears() {  
 return gears;  
 }  
  
 public void setGears(int gears) {  
 this.gears = gears;  
 }  
}

public class Tesla extends Car {  
 private int range;  
  
 public Tesla(int numberOfDoors, int gears, int range) {  
 super(numberOfDoors, gears);  
 this.range = range;  
 }  
  
 public void charge(int power) {  
 this.range += power;  
 }  
  
 public int getRange() {  
 return range;  
 }  
  
 public void setRange(int range) {  
 this.range = range;  
 }  
}

public class ModelS extends Tesla {  
 private int horsePower;  
  
 public ModelS(int numberOfDoors, int gears, int range, int horsePower) {  
 super(numberOfDoors, gears, range);  
 this.horsePower = horsePower;  
 }  
  
 public void goFaster(int powerUp) {  
 this.horsePower += powerUp;  
 }  
  
 public int getHorsePower() {  
 return horsePower;  
 }  
  
 public void setHorsePower(int horsePower) {  
 this.horsePower = horsePower;  
 }  
}

# Interfaces

Utwórz nowy interfejs, który będzie zaimplementowany w klasie ModelS i będzie zawierał metodę goFaster

public interface SportCarInterface {  
 void goFaster(int powerUp);  
}

public class ModelS extends Tesla implements SportCarInterface{

…

}

# Abstracts

Zmien klasy Car oraz Tesla na abstrakcyjne. Klasie Car dodaj metodę drive z domyślną implementacją, natomiast klasie Tesla dodaj abstrakcyjna metode lowPower

public abstract class Car {  
 private int numberOfDoors;  
 private int gears;  
  
 public Car(int numberOfDoors, int gears) {  
 this.numberOfDoors = numberOfDoors;  
 this.gears = gears;  
 }  
  
 public void drive(){  
 System.*out*.println("Im driving");  
 }  
  
 public int getNumberOfDoors() {  
 return numberOfDoors;  
 }  
  
 public void setNumberOfDoors(int numberOfDoors) {  
 this.numberOfDoors = numberOfDoors;  
 }  
  
 public int getGears() {  
 return gears;  
 }  
  
 public void setGears(int gears) {  
 this.gears = gears;  
 }  
}

public abstract class Tesla extends Car {  
 private int range;  
  
 public Tesla(int numberOfDoors, int gears, int range) {  
 super(numberOfDoors, gears);  
 this.range = range;  
 }  
  
 public abstract void lowPower();  
  
 public void charge(int power) {  
 this.range += power;  
 }  
  
 public int getRange() {  
 return range;  
 }  
  
 public void setRange(int range) {  
 this.range = range;  
 }  
}

public class ModelS extends Tesla implements SportCarInterface{  
 private int horsePower;  
  
 public ModelS(int numberOfDoors, int gears, int range, int horsePower) {  
 super(numberOfDoors, gears, range);  
 this.horsePower = horsePower;  
 }  
  
 public void goFaster(int powerUp) {  
 this.horsePower += powerUp;  
 }  
  
 public int getHorsePower() {  
 return horsePower;  
 }  
  
 public void setHorsePower(int horsePower) {  
 this.horsePower = horsePower;  
 }  
  
 @Override  
 public void lowPower() {  
 System.*out*.println("Im low, charge me");  
 }  
}

# Try catch

Utwórz nową NoPowerException która będzie rzucana przez ModelS gdy jego range spadnie poniżej 0. Następnie spróbuj wywołać metodę lowPower tak, aby został rzucony wyjątek i obsłuż go za pomocą try catch.

@Override  
public void lowPower() {  
 if (getRange() <= 0) {  
 System.*out*.println("Im low, charge me");  
 throw new NoPowerException();  
 }  
}

public void testLowPower() {  
 ModelS modelS = new ModelS(4, 6, 0, 700);  
 try {  
 modelS.lowPower();  
 } catch (NoPowerException ex){  
 modelS.charge(100);  
 }  
}

Zjazd 3

Do pracy przy zjeździe 3 potrzebny jest prostu projekt, w którym posiadamy klasę Main, klasę modelową oraz klasę serwisową, w której utworzymy różne metody potrzebne do pracy na wymyślonym przez nas obiekcie. Za model w naszym projekcie przyjęliśmy klasę Pizza.

# Lambda

Utwórz metodę, która przyjmie Listę integerów z id’kami pizz, następnie przemapuj ją do Listy obiektów typu Pizza.

public void mapPizzaIdToPizza(List<Integer> pizzasIds) {  
 List<Pizza> collect = pizzasIds.stream().map(id -> findById(id)).collect(Collectors.*toList*());  
}

# Optional

Napisz metodę, która przy wykorzystaniu Optional zwróci (bądź nie, jeżeli nie znajdzie) nam obiekt o wpisanym przez nas id.

public Optional<Pizza> findById(int id) {  
 return pizzaList.stream().filter(pizza -> pizza.getId() == id).findFirst();  
}

# Method reference

Zamień wyrażenie Lambda na Method reference.

pizzasIds.stream().map(this::findById)

Zjazd 4

# Simple pub/pizzeria project

Napisz prosty projekt do obsługi pubu/pizzerii i udostępnij kod na GIT. Aktualne rozwiązania dostępne <https://github.com/sughalith/pjatk_car_factory> na odpowiednich branchach

# Conflicts on GIT

Na wyżej utworzonym projekcie doprowadź do konfliktu ze zmianami dowolnej osoby z zespołu.