OOP with Java

Homework 03: Classes and Objects

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://iao.hfuu.edu.cn

Hefei University, South Campus 2	合肥学院 南艳湖校区/南2区
Faculty of Computer Science and Technology	计算机科学与技术系
Institute of Applied Optimization	应用优化研究所
230601 Shushan District, Hefei, Anhui, China	中国 安徽省 合肥市 蜀山区 230601
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99	经济技术开发区 锦绣大道99号

Outline

1. Introduction
2. Tasks

website

Introduction

- We want to practice working with objects and classes
- We use all the stuff we have learned before, including expressions, if-then-else, loops, static routines from `java.util.Math`, etc.
- This homework is comprised of two task
- Send me a zip archive named `hw03_[your_student_id].zip` (where `[your_student_id]` is replaced with your student id) with one answer-folder for each homework task (names `hw03-1` and `hw03-2`)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
   1. with a member variable `double sideLength` to hold the side lengths,
   2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
   1. with a member variable `double sideLength` to hold the side lengths,
   2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
   3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for:
    1. equilateral triangles (`http://en.wikipedia.org/wiki/Equilateral_triangle`)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for:
    1. equilateral triangles (`http://en.wikipedia.org/wiki/Equilateral_triangle`)
    2. squares (`http://en.wikipedia.org/wiki/Square`)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for:
    1. equilateral triangles (http://en.wikipedia.org/wiki/Equilateral_triangle)
    2. squares (http://en.wikipedia.org/wiki/Square)
    3. regular pentagons (http://en.wikipedia.org/wiki/Pentagon#Regular_pentagons)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for:
    1. equilateral triangles (http://en.wikipedia.org/wiki/Equilateral_triangle)
    2. squares (http://en.wikipedia.org/wiki/Square)
    3. regular pentagons (http://en.wikipedia.org/wiki/Pentagon#Regular_pentagons)
    4. regular hexagons (http://en.wikipedia.org/wiki/Hexagon#Regular_hexagon)

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for:
    1. equilateral triangles (http://en.wikipedia.org/wiki/Equilateral_triangle)
    2. squares (http://en.wikipedia.org/wiki/Square)
    3. regular pentagons (http://en.wikipedia.org/wiki/Pentagon#Regular_pentagons)
    4. regular hexagons (http://en.wikipedia.org/wiki/Hexagon#Regular_hexagon)
4. Create a `Main` class which instantiates each of these classes and prints the area of the corresponding polygons with `sideLength` 1

Task hw03-1: Class Hierarchy I

1. Develop a class hierarchy for convex, simple, regular, equilateral, equiangular polygons (with sides all having the same length and angles being the same).
2. There should be a base class `Polygon`:
    1. with a member variable `double sideLength` to hold the side lengths,
    2. with a one-parameter constructor taking a corresponding parameter and initializing the above member variable,
    3. with a method `int numberOfSides()` to return the actual number of sides of the polygon (returning `0` in this base class, to be overridden by subclasses)
    4. with a method `double area()` returning the inside area of the polygon (returning `0` in this base class, to be overridden by subclasses)
3. Create suitable sub-classes of `Polygon` implementing the methods for:
    1. equilateral triangles (http://en.wikipedia.org/wiki/Equilateral_triangle)
    2. squares (http://en.wikipedia.org/wiki/Square)
    3. regular pentagons (http://en.wikipedia.org/wiki/Pentagon#Regular_pentagons)
    4. regular hexagons (http://en.wikipedia.org/wiki/Hexagon#Regular_hexagon)
4. Create a `Main` class which instantiates each of these classes and prints the area of the corresponding polygons with `sideLength` 1
5. The answer-folder for this task contains the complete Eclipse project, including source code (.java) and compiled (.class) file.

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
    3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd(\texttt{numerator}, \texttt{denominator}) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (`https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations`))

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
   1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
   2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
   3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd($ `numerator` , `denominator` $) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (`https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations`))
   4. if the fraction is negative, the sign is stored in the `numerator` , i.e., -0.2 be $\frac{-1}{5}$, not $\frac{1}{-5}$ (the constructor must take care of this)

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (https://en.wikipedia.org/wiki/Fraction_(mathematics))
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
    3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd($ `numerator` , `denominator` $) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations))
    4. if the fraction is negative, the sign is stored in the `numerator` , i.e., -0.2 be $\frac{-1}{5}$, not $\frac{1}{-5}$ (the constructor must take care of this)
    5. overrides the inherited methods `toString()` , `doubleValue()` , and `floatValue()` with reasonable behavior

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
    3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd($ `numerator` , `denominator` $) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (`https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations`))
    4. if the fraction is negative, the sign is stored in the `numerator` , i.e., -0.2 be $\frac{-1}{5}$, not $\frac{1}{-5}$ (the constructor must take care of this)
    5. overrides the inherited methods `toString()` , `doubleValue()` , and `floatValue()` with reasonable behavior
    6. overrides the inherited methods `intValue()` and `longValue()` to return $\left\lfloor \frac{\texttt{numerator}}{\texttt{denominator}} \right\rfloor$

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
    3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd($ `numerator` , `denominator` $) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (`https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations`))
    4. if the fraction is negative, the sign is stored in the `numerator` , i.e., -0.2 be $\frac{-1}{5}$, not $\frac{1}{-5}$ (the constructor must take care of this)
    5. overrides the inherited methods `toString()` , `doubleValue()` , and `floatValue()` with reasonable behavior
    6. overrides the inherited methods `intValue()` and `longValue()` to return $\left\lfloor\frac{\texttt{numerator}}{\texttt{denominator}}\right\rfloor$
    7. implements the instance methods `add` , `sub` , `mul` , `div` , and `mod` to return the results of the addition, subtraction, multiplication, division, and rest of the division of the current number and their one argument of type `Fraction`

Task hw03-2: Class Hierarchy II

1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (`https://en.wikipedia.org/wiki/Fraction_(mathematics)`)
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
    3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd($ `numerator` , `denominator` $) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (`https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations`))
    4. if the fraction is negative, the sign is stored in the `numerator` , i.e., -0.2 be $\frac{-1}{5}$, not $\frac{1}{-5}$ (the constructor must take care of this)
    5. overrides the inherited methods `toString()` , `doubleValue()` , and `floatValue()` with reasonable behavior
    6. overrides the inherited methods `intValue()` and `longValue()` to return $\left\lfloor\frac{\texttt{numerator}}{\texttt{denominator}}\right\rfloor$
    7. implements the instance methods `add` , `sub` , `mul` , `div` , and `mod` to return the results of the addition, subtraction, multiplication, division, and rest of the division of the current number and their one argument of type `Fraction`
5. Create a `Main` class which computes and prints the result of $\frac{\frac{16}{3}*(\frac{2}{3}-\frac{10}{70})}{\frac{63}{176}}$ both as fraction and as `double` (you can verify your results with tools such as `http://www.calculator.net/fraction-calculator.html`)

Task hw03-2: Class Hierarchy II

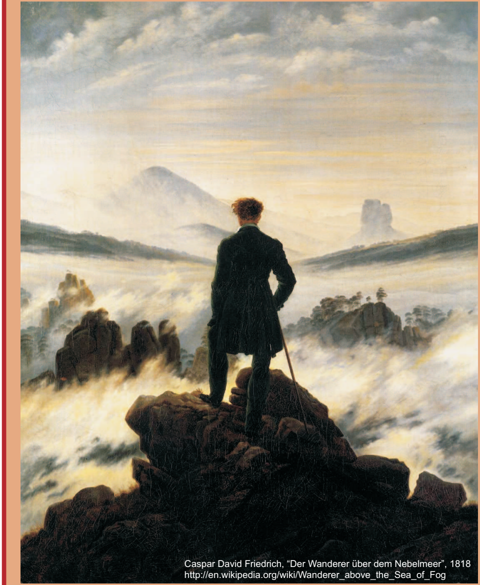
1. We want to extend the class `java.lang.Number` by creating a new sub-class for dealing with *fractions*
2. A fraction is a number like $\frac{3}{4}$, which equals 0.75
3. In other words, a number $\frac{i}{j}$ where $i, j \in \mathbb{Z}$ (https://en.wikipedia.org/wiki/Fraction_(mathematics))
4. Create a class named `Fraction` which:
    1. has two member variables of type `long` named `numerator` and `denominator` such that its instances represent numbers $\frac{\texttt{numerator}}{\texttt{denominator}}$
    2. has two constructors, one which takes a single number i to represent `numerator` $= i$ and `denominator` $= 1$ and one which accepts the values of both `numerator` and `denominator`
    3. both the `numerator` and `denominator` shall always be normalized by using the greatest common divisor, i.e., $gcd($`numerator`, `denominator`$) \overset{!}{=} 1$ (the constructor must take care of this by dividing both input parameters by their gcd, which you can compute using, e.g., Euclid's algorithm (https://en.wikipedia.org/wiki/Euclidean_algorithm#Implementations))
    4. if the fraction is negative, the sign is stored in the `numerator`, i.e., -0.2 be $\frac{-1}{5}$, not $\frac{1}{-5}$ (the constructor must take care of this)
    5. overrides the inherited methods `toString()`, `doubleValue()`, and `floatValue()` with reasonable behavior
    6. overrides the inherited methods `intValue()` and `longValue()` to return $\left\lfloor \frac{\texttt{numerator}}{\texttt{denominator}} \right\rfloor$
    7. implements the instance methods `add`, `sub`, `mul`, `div`, and `mod` to return the results of the addition, subtraction, multiplication, division, and rest of the division of the current number and their one argument of type `Fraction`
5. Create a `Main` class which computes and prints the result of $\frac{\frac{16}{3}*(\frac{2}{3}-\frac{10}{70})}{\frac{63}{176}}$ both as fraction and as `double` (you can verify your results with tools such as http://www.calculator.net/fraction-calculator.html)
6. The answer-folder for this task contains the complete Eclipse project, including source code (.java) and compiled (.class) file.

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://iao.hfuu.edu.cn

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China



Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog