2. Java Basics

4. Java Classes

Class – why?

- Classes split application code to parts (from sophisticated to simple)
- Very often class is a model of an object from the real world
- Java says: Everything is an object
- Class describes object behaviour
- Class is a type

Class Description

class *name {*

// field declarations

// method declarations

}

Class Fields

- Class fields should be declared inside class out of all class methods
- Fields can have primitive type, or reference type such as array or object
- Fields are visible to all instance methods
- Fields are automatically initialized (reference types with null, number types with zero, boolean – with false)

Defining Methods

return_type *method_name* (*parameter_list*){
  // method body
}

Example:

```
int getFinalData(int a, int r){
    int b = r % 18;
    return a * 2 + b;
}
```

Return Type

- The return type describes the value that comes back from the method
- A method can have **void** return type
- Any method that is not declared void must contain a return statement with a corresponding return value
- Return statements for void return type is not necessary

Parameters

- Any data type is possible for a parameter of a method
- Construct ***varargs*** is used to pass an arbitrary number of values (e.g. *type*... args)
- Varargs can be used *only* in the final argument position
- Parameters are passed into methods *by value*.
- The values of the object's fields *can* be changed in the method

Constructors

- Constructor name should be the same as class name
- Constructor has no return type
- The compiler automatically provides a no-argument, default constructor for any class without parameters – **don't use this possibility, declare such constructor explicitly**
- A class can have several constructors (with different sets of parameters)

Objects

- Creating Object:

  class_name object_variable **=** **new** *construtor_call;*

- Declaring a Variable to Refer to an Object:

  class_name object_variable;

- Calling an Object's Methods:

  object_variable.methodName(argumentList);

Using the this Keyword

- **this** is a reference to the *current object*
- The most common example:

```
class Point {
    int x = 0;
    int y = 0;

    //constructor
    Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

Complex Numbers (1 of 4)

- Is it always possible to solve square equation $ax^2 + bx + c = 0$ within real numbers set?

Complex Numbers (2 of 4)

- Is it always possible to solve square equation $ax^2 + bx + c = 0$ within real numbers set?
- No, if $b^2 - 4ac < 0$ it is impossible.
- We can expand real number set to complex number set introducing new number type - complex unit *i* - in such a way:

  i * *i* = -1

Complex Numbers (3 of 4)

- Number of *a* + *b* * *i* type where *a* and *b* are real is called complex number.
- Every square equation can be solved within complex numbers set.
- Moreover, every algebraic equation (with arbitrary power) always can be solved within complex numbers set.

Complex Numbers (4 of 4)

- To add complex numbers use formula

$$(a_1 + b_1 i) + (a_2 + b_2 i) = (a_1 + a_2) + (b_1 + b_2)i$$

- To multiply complex numbers use formula

$$(a_1 + b_1 i) * (a_2 + b_2 i) = (a_1 a_2 - b_1 b_2) + (a_1 b_2 + a_2 b_1)i$$

- To find absolute value of complex number use formula $r = \sqrt{a^2 + b^2}$

Exercise 2.4.1.

- Create a class for saving and manipulating complex numbers.

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and accessors
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and getters/setters
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Class for Complex Numbers

```
/**
 * Represents complex numbers
 */
class Complex {

}
```

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and getters/setters
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Class for Complex Numbers

```
class Complex {
    /**
     * Real part of a complex number
     */
   double r;
    /**
     * Imaginary part of a complex number
     */
    double im;
}
```

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and getters/setters
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Class for Complex Numbers

```
/**
 * Default constructor sets complex zero
 */
Complex(){
 r = 0.0;
 im = 0.0;
}
```

Class for Complex Numbers

```
/**
 * Initializes a complex number
 * @param r - real part of a complex number
 * @param im - imaginary part of a complex
number
 */
Complex(double r, double im){
    this.r = r;
    this.im = im;
}
```

Class for Complex Numbers

```
double getR() { return r; }
void setR(double value){ r = value; }

double getIm() { return im; }
void setIm(double value){ im = value; }
```

Accessors in Eclipse

- Right click in text editor, and select Source > Generate Getters and Setters .. Menu item
- Check necessary boxes for creating getters and / or setters, select access modifiers and click Ok button.

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and getters/setters
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Class for Complex Numbers

```
/**
 * Returns module of the complex number
 */
double getModule(){
}
```

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and getters/setters
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Unit Test for getModule I

```
public class E241TestComplex {
   public static void main(String[] args) {
    Complex test1 = new Complex();
    test1.setR(3.0);
    test1.setIm(4.0);
    System.out.println("module = " +
   test1.getModule());
   }
}
```

Unit Test for getModule II

```
public class E241TestComplex {
    public static void main(String[] args) {
     Complex test1 = new Complex();
     test1.setR(3.0);
     test1.setIm(4.0);
    double res = test1.getModule();
    if (res == 5.0){
     System.out.println("getModule test is true");}
    else{
     System.out.println("getModule test failed");}
    }
}
```

Step by Step Solution

1. Check problem definition. If it is clear go to step 2
2. Create class
3. Describe class fields
4. Create constructors and getters/setters
5. Create method prototypes
6. Create unit tests
7. Create method bodies

Class for Complex Numbers

```
/**
 * Returns module of the complex number
 */
double getModule(){
 return Math.sqrt(r * r + im * im);
}
```

Class for Complex Numbers

```
Complex add(Complex value){
}

Complex multiply(Complex value){
}
```

Unit Test for getModule

```
public class E241TestComplex {
    public static void main(String[] args) {
     Complex conjugate1 = new Complex(3.0, 2.0);
     Complex conjugate2 = new Complex(3.0, -2.0);
     Complex result = conjugate1.add(conjugate2);
    r = result.getR(); im = result.getIm();
    if ((r == 13.0) && (im == 0.0)){
     System.out.println("multiply test 1 is true");}
    else{ System.out.println("multiply test 1 failed");}
    }
}
```

Class for Complex Numbers

```
Complex add(Complex value){
   return new Complex(this.r + value.getR(), this.im + value.getIm());
}
Complex multiply(Complex value){
   double rr = this.r * value.getR() - this.im *     value.getIm();
   double rim = this.r * value.getIm() + this.im * value.getR();
   return new Complex(rr, rim);
}
```

Exercise 2.4.1.

- See 241Complex project for the full text

Exercise 2.4.2 (1 of 2)

- 3D Vector is ordered sequence of 3 numbers (vector's coordinates):

$$\bar{r} = (x_1, x_2, x_3)$$

- Sum of two vectors

$$\bar{r}_1 = (x_{11}, x_{12}, x_{13}) \qquad \bar{r}_2 = (x_{21}, x_{22}, x_{23})$$

is a vector

$$\bar{r} = \bar{r}_1 + \bar{r}_2 = (x_{11} + x_{21}, x_{12} + x_{22}, x_{13} + x_{23})$$

Exercise 2.4.2 (2 of 2)

- Scalar product of two vectors

$$\bar{r}_1 = (x_{11}, x_{12}, x_{13}) \qquad \bar{r}_2 = (x_{21}, x_{22}, x_{23})$$

is a number

$$p = x_{11}x_{21} + x_{12}x_{22} + x_{13}x_{23}$$

- Vector product of two vectors is a vector

$$\bar{r} = \bar{r}_1 \times \bar{r}_2 = (x_{12}x_{23} - x_{13}x_{22}, x_{13}x_{21} - x_{11}x_{23}, x_{11}x_{22} - x_{12}x_{21})$$

- Vector's module is a number

$$m = \sqrt{x_1^2 + x_2^2 + x_3^2}$$

- You should create a class Vector3D for vector saving and manipulating