Features Explore Pricing

This repository Search

Sign in or Sign up

havanagrawal / c2c2017

Watch 1 Star 2 Fork 2

Code Issues 0 Pull requests 0 Projects 0 Pulse Graphs

Branch: master ▾ c2c2017 / Session03 /

Create new file Find file History

havanagrawal Add sample code for Strings and static modifier Latest commit 0e00b92 5 days ago

..

README.md	Add notes for char datatype	5 days ago
StaticVariableExampleRunner.java	Add sample code for Strings and static modifier	5 days ago
StringMethods.java	Add sample code for Strings and static modifier	5 days ago

README.md

Session 3

Table of Contents

Function Overloading

Function Overloading simply means defining two or more functions with the same name, but with distinguishable signatures.

Function overloading is done by changing either:

1. Types of the arguments
2. Number of arguments

The return type is *NOT* part of the signature, and so changing the return type is not sufficient for overloading.

```
// Two arguments
public static int min(int a, int b) {
    return a < b ? a : b;
}

// Three arguments
```

```
public static int min(int a, int b, int c) {
    return min(min(a, b), d);
}

// Single argument
public static int min(int[] arr) {
    int min = arr[0];

    for (int i = 1; i < arr.length; i++) {
        min = arr[i] < min ? arr[i] : min;
    }

    return min;
}

// Two arguments, but different type
public static double min(double a, double b) {
    return Math.min(a, b);
}

// This will NOT compile, because you cannot overload by return type
public static long min(int a, int b) {
    return (long)(a < b ? a : b);
}
```

Practice:

1. Write an overloaded function for GCD (Greatest Common Divisor) with the following signatures:
    i. `int gcd(int a, int b)`
    ii. `int gcd(int a, int b, int c)`
    iii. `int gcd(int[] arr)`
2. The data type of `System.out` is PrintStream. The `println` method belongs to this instance. Check out the docs and look for the `println` method, and its different overloaded variations.

Classes

A class is like a blueprint for creating instances of an entity. It defines the member variables and methods for those instances.
For example, a `Person` class can define instance variables such as `name`, `age`, `weight`, and instance methods `growOld`, `eatFood`. Notice how instance methods are verb-like, and instance variables are properties of a person.

```
class Person {
    String name;
    int age;
    double weightInKg;

    void growOld() {
        age = age + 1;
    }

    void eatFood() {
        weightInKg = weightInKg + 0.1;
    }
}
```

Instance methods are non-static methods. This means that they belong to each instance of a class, and not to the class as a whole. This makes complete sense in the above class, each Person instance can independently eat food, and grow old.

Constructors

Constructors are a special variant of functions. They can only be used with the `new` keyword, to create an instance of a class.

If no constructor is defined, Java provides you with a 0-argument default constructor. In the above Person example, since we have not defined one, it is

perfectly valid to create an instance by:

```
Person p = new Person();
```

However, this is clearly not what we want. We'd prefer instantiating the Person with his/her name, age and weight. So we can define a constructor as:

```
class Person {
  ...

  Person(String n, int a, double w) {
    name = n;
    age = a;
    weight = w;
  }
}
```

Remember that, since constructors are functions, you can overload them as well.

```
class Person {
  ...

  Person(String n) {
    name = n;
    age = 20;      // some default value
    weight = 75;   // some default value
  }

  Person(String n, int a, double w) {
    name = n;
    age = a;
    weight = w;
  }
}
```

Instance Variables

An instance variable is a property of a single instance of a class.

When you declare an instance variable, it is given a default value. The default values are as follows:
Numeric data types (byte, short, int, long, float, double): *0*
Boolean: *false*
Object references: *null* (We will see this in the next session)

However, you may want to provide your own default values. You can do so by assigning a default value to the instance variable at the point of declaration.

```
class Person {
  int age = 20;
  double weight;  // Defaults to 0.0
  String name;    // Defaults to null
}
```

Instance Methods

An instance method is used to modify or access one or more instance variables. In the `Person` class, `growOld` modifies the age property of that instance. So for example:

```
public static void main(String[] args) {
  Person p1 = new Person("Brandon", 16, 45.0);
```

```
    Person p2 = new Person("Rob", 21, 75.2);

    System.out.println(p1.age);          // Prints 16
    System.out.println(p1.weight);       // Prints 45.0

    System.out.println(p2.age);          // Prints 21
    System.out.println(p2.age);          // Prints 75.2

    // It is Brandon's birthday! :P
    p1.growOld();

    // Coz he's hungry
    p2.eatFood();

    System.out.println(p1.age);          // Prints 17
    System.out.println(p1.weight);       // Prints 45.0

    System.out.println(p2.age);          // Prints 21
    System.out.println(p2.weight);       // Prints 75.3
}
```

Static Methods and Instances

Sometimes, certain values are properties of the class itself, and not different for each instance. For `Person`, you could say that the number of bones are 206 (conveniently ignoring other factors).

```
class Person {
    static int noOfBones = 206;
}
```

Static variables are accessed by using the class name:

```
// There are two examples of static variables in the below line, can you see it?
System.out.println(Person.noOfBones);
```

Similarly, static methods belong to the class, and not for each individual instance. Static methods are much more common than static variables, and we have already seen quite a few of them; `Math.min()`, `System.getCurrentTimeMillis()`, `main`.

While it is not recommended, you *can* access static methods and variables by using instances of the class as well:

```
// Just an example, don't do this.
Person p = new Person();
System.out.println(p.noOfBones);
```

This makes sense because it is a property that belongs to the class itself, and so logically applies to every instance of the class.

HOWEVER, it is *NOT* possible to access instance variables and methods from a "static context", i.e. from a static method.

```
class Person {
    int age = 0;

    static int someMethod() {
        age += 1;            // THIS WILL NOT COMPILE
    }
}
```

This is obvious when you think about it. Since a static method belongs to the class as a whole, it does not have a concept of age or weight. It makes no sense to increment age for the Person class, while it did make sense to increment age for an *instance* of the class.

Practice

1. Write a class PhoneEmulator. It should have the following methods:
    i. call(long phoneNumber)
    ii. hangUp()
    iii. isCallOngoing()
    iv. redial()
    v. toggleMute()
    vi. isMuted()
    vii. getBill(int rateScheme)
        a. 0 = 2 paisa per second, constant
        b. 1 = free for first 10 seconds, then 3 paisa/s
        c. 2 = 1 paisa/s for first 20 seconds, then 2 paisa/s
        Decide on appropriate return types.
        Think about edge cases.
        a. What happens if you try to hang up when there is no ongoing call?
        b. What happens if you invoke call, when there is already an ongoing call?
        Helpful method: System.getCurrentTimeMillis
2. Write a class 2DPoint. It should have the following constructors and methods
    i. Point()
    ii. Point(double x, double y)
    iii. euclideanDistanceTo(Point p) - Distance to Point p, using Euclidean Distance Formula
    iv. manhattanDistanceto(Point p) - Distance to Point p using Manhattan Distance Formula
    v. midpoint(Point a, Point b) - Return the midpoint of the two points.
    vi. A static variable called ORIGIN, that represents the origin.
    Again, think about which of these methods should be static.

The final modifier

The final modifier can be applied to

1. Variables
2. Methods (next session)
3. Classes (next session)

When the final modifier is applied to a variable, it means that once the variable is assigned a value, it cannot be modified.

```
class Student {
    // This is formally known as a "blank final".
    // If you do not initialize this in every constructor, you will get a compile time error
    final int rollNo;

    Student(int r) {
        rollNo = r;
    }
}
```

You can assign a value to a final instance variable

1. In the constructor
2. The point of declaration
3. Instantiation block (next session)

Remember that when you assign the value at the point of declaration, EVERY instance of the class will have that value, and can not be changed at all.

```
class Person {
    // just an example
    final int noOfBones = 206;
```

```
}
```

The final modifier can also be used in conjunction with the static modifier. In such cases, the variable is as good as a constant, and are named with the conventions of ALL_CAPS_WITH_UNDERSCORES. A great example is Math.PI. Other examples:

```
class Constants {
    static final int NO_OF_SECONDS_IN_ONE_MINUTE = 60;
    static final int NO_OF_HOURS_IN_ONE_DAY = 24;

    // Convenient way of using this number in competitive programming
    static final long MOD = 1000_000_000 + 7;

    // Throwback :P
    static final double AVOGADRO_CONSTANT = 6.022E23;
}
```

You can initialize a value to static final variable

1. At the point of declaration
2. In the static block (next session)

Practice:

1. Revisit these problems. Think whether any of your instance variables should be declared as final or not.

The String class

The String class is a rather special class in Java, since it has several behaviors that make it appear to be a primitive.

```
// Use of a String literal
String a = "hello";

// The + operator can be used on strings
String b = a + ", world";
```

The best place to understand a new class is by referring to the Javadocs. The JavaDocs for the String class are here

Working with the char datatype

When manipulating String instances, it is often useful to be able to manipulate individual characters. A useful tip to remember is that char is a numeric-like data type, and you can do arithmetic on it.

```
// You can iterate!
for (char c = 'a'; c <= 'z'; c++) {
    // Do something with c
}

// To convert a lowercase char into uppercase
int upperLowerDiff = 'A' - 'a';
char bigH = (char)('h' + upperLowerDiff);

// You can reason about this very simply
// 'A' - 'a' = 'H' - 'h'
// 'A' - 'a' + 'h' = 'H'

// Similarly

// To convert uppercase char into lowercase
int lowerUpperDiff = 'a' - 'A';
char bigH = (char)('H' + upperLowerDiff);
```

```
// When you want to count frequencies of characters, a nifty trick is to use an array, and index it by character value
String s = "wewillcountcharacters";
char[] cArray = s.toCharArray();

// Given that you will MOSTLY work with only ASCII, and not UNICODE
// You only need a size of 128 for counting characters in an arbitrary String
int[] frequencies = new int[128];

for (char c : cArray) {
    frequencies[c] += 1;
}
```

Note: Arrays are object-like. You can notice several similarities between arrays and objects:

1. The .length property
2. Typicaly use of `new` when creating arrays.

This gives us a third feature of an array:
When you don't assign values to individual cells in an array, they follow the default value rule.
This is how we are able to execute the line `frequencies[c] += 1`, because all cells of the array were initialized to 0.

Practice:

1. https://www.hackerrank.com/challenges/java-strings-introduction
2. https://www.hackerrank.com/challenges/java-substring
3. https://www.hackerrank.com/challenges/java-string-compare
4. https://www.hackerrank.com/challenges/caesar-cipher-1
5. https://www.hackerrank.com/challenges/camelcase

Assignments

HackerRank

1. https://www.hackerrank.com/challenges/java-string-reverse
2. https://www.hackerrank.com/challenges/java-anagrams
3. https://www.hackerrank.com/challenges/mars-exploration
4. https://www.hackerrank.com/challenges/funny-string
5. https://www.hackerrank.com/challenges/gem-stones
6. https://www.hackerrank.com/challenges/alternating-characters
7. https://www.hackerrank.com/challenges/pangrams
8. https://www.hackerrank.com/challenges/the-love-letter-mystery
9. https://www.hackerrank.com/challenges/game-of-thrones
10. https://www.hackerrank.com/challenges/two-strings
11. https://www.hackerrank.com/challenges/string-construction

Miscellaneous

1. Make a `MathUtil` class. It should contain the following methods:
    i. `gcd(long a, long b)`
    ii. `lcm(long a, long b)`
    iii. `min(long[] arr)`
    iv. `min(int[] arr)`
    v. `max(long[] arr)`
    vi. `max(int[] arr)`
    vii. `average(long[] arr)`
    Think whether these should be instance methods or static methods. Decide on appropriate data types.

© 2017 GitHub, Inc. Terms Privacy Security Status Help

Contact GitHub API Training Shop Blog About