Java 是否支持默认参数值?

问 12 年零 3 个月前　活跃 4 天前　浏览了 150 万次

1881年

350

我遇到了一些具有以下结构的 Java 代码：

```
public MyParameterizedFunction(String param1, int param2)
{
    this(param1, param2, false);
}

public MyParameterizedFunction(String param1, int param2, boolean param3)
{
    //use all three parameters here
}
```

我知道在 C++ 中我可以为参数分配一个默认值。例如：

```
void MyParameterizedFunction(String param1, int param2, bool param3=false);
```

Java 是否支持这种语法？有什么理由为什么这两个步骤的语法更可取吗？

爪哇　方法　参数　超载　默认参数

分享　改进这个问题　跟随

19 年 5 月 8 日 9:01编辑

狼
8,813　7　50　94

问09 年 6 月 15 日 18:04

纳维
22.6k　7　19　11

100　不会。但是，Builder 模式可以提供帮助。 —— 戴夫·贾维斯 2010 年 5 月 27 日 2:58

76　我真的很想念这个功能。在修改现有代码以将额外参数传递给函数或构造函数时，它有很大帮助 —— 贾廷 2015 年 1 月 1 日 7:25

4　@Jatin 通过 Eclipse“更改方法签名”重构，您可以添加一个参数并提供现有调用者将使用的默认值。 —— 埃尔文·博尔维特 2015 年 1 月 12 日 15:17

2　@ErwinBolwidt 谢谢。我使用的是 Android Studio，它还可以选择重构方法并提供默认值。相当有用。 —— 贾廷 2015 年 1 月 14 日 11:40

3　@temporary_user_name `public MyParameterizedFunction(String param1, int param2)` 是构造函数，而不是方法声明。 —— 马里奥·伊沙克 2019-01-24 22:49

26 个回答

积极的　最老的　投票

不，您找到的结构是 Java 处理它的方式（即，使用重载而不是默认参数）。

助。这是当你有足够的复杂性时，区分是困难的。一个明确的情况是您必须使用参数的顺序进行区分，而不仅仅是数字和类型。

分享 改进这个答案 Follow

edited Sep 13 '13 at 17:12

Peter Mortensen
28.9k 21 96 123

answered Jun 15 '09 at 18:14

Kathy Van Stone
23.9k 3 30 40

29 this was in 2009, `static` anything is pretty much considered harmful in 2015. Type safe fluent `Builder` instances that enforce complete and valid construction contracts are a much better solution now. – user177800 Dec 18 '15 at 4:27

165 @JarrodRoberson: Static factory methods are no more harmful than `new`. They are used all the time in new code. Builders for simple value objects are often the result of over-engineering. – Lii Jan 22 '16 at 8:38

13 @JarrodRoberson: Interesting way of forcing correct use via the compiler, thanks for sharing! Friendly suggestion for future posts: 300 lines of uncommented source code is probably a bit much to digest for most people (code is harder to read than to write, after all). Thanks again! – Christian Aichinger Jan 27 '16 at 21:55

17 @JarrodRoberson: Nice, looking forward to it! What I wanted to communicate: as a reader of your blog, a 50 line example with a brief text description of what's going on would help me more than 300 lines without context. – Christian Aichinger Jan 27 '16 at 22:12

13 @user177800 Disagree - static methods if written as pure functions are perfectly fine. It's when a static function mutates state that they become a problem... – Levi Fuller Dec 18 '18 at 3:47

712

No, but you can use the Builder Pattern, as described in this Stack Overflow answer.

As described in the linked answer, the Builder Pattern lets you write code like

```
Student s1 = new StudentBuilder().name("Eli").buildStudent();
Student s2 = new StudentBuilder()
                .name("Spicoli")
                .age(16)
                .motto("Aloha, Mr Hand")
                .buildStudent();
```

in which some fields can have default values or otherwise be optional.

Share Improve this answer Follow

edited May 23 '17 at 12:34

Community Bot
1 1

answered Jun 15 '09 at 19:20

Eli Courtwright
169k 63 204 255

158 Finally e great less-than-2-pages example of the Builder pattern. – nevvermind Nov 8 '12 at 21:38

17 Im curious though, why do we need a builder class when using the builder pattern. I was thinking of Student s1 = new Student().name("Spicolo").age(16).motto("Aloha, Mr Hand); – ivanceras Feb 14 '13 at 8:32

61 @ivanceras: It's relevant when classes have required fields, and you don't want to be able to

67 @ivanceras: another reason is you might want your class to be immutable after construction, so you wouldn't want methods in it that change its values. – Jules Feb 27 '13 at 16:02

5 @ivanceras: I used Builders for 3 things - eliminating the multiple argument and fluent initialization, immutability and most importantly I feel to validate the domain object in the build() method.Why create an object instance if is invalid.You can also overload static factory methods like in the above case buildFreshman(), buildSenior() etc – Abhijeet Kushe Aug 16 '15 at 22:00

553

There are several ways to simulate default parameters in Java:

1. **Method overloading.**

```
void foo(String a, Integer b) {
    //...
}

void foo(String a) {
    foo(a, 0); // here, 0 is a default value for b
}

foo("a", 2);
foo("a");
```

One of the limitations of this approach is that it doesn't work if you have two optional parameters of the same type and any of them can be omitted.

2. **Varargs.**

a) All optional parameters are of the same type:

```
void foo(String a, Integer... b) {
    Integer b1 = b.length > 0 ? b[0] : 0;
    Integer b2 = b.length > 1 ? b[1] : 0;
    //...
}

foo("a");
foo("a", 1, 2);
```

b) Types of optional parameters may be different:

```
void foo(String a, Object... b) {
    Integer b1 = 0;
    String b2 = "";
    if (b.length > 0) {
      if (!(b[0] instanceof Integer)) {
          throw new IllegalArgumentException("...");
      }
      b1 = (Integer)b[0];
    }
    if (b.length > 1) {
        if (!(b[1] instanceof String)) {
            throw new IllegalArgumentException("...");
        }
        b2 = (String)b[1];
        //...
    }
    //
```

```
foo("a", 1);
foo("a", 1, "b2");
```

The main drawback of this approach is that if optional parameters are of different types you lose static type checking. Furthermore, if each parameter has different meaning you need some way to distinguish them.

3. **Nulls.** To address the limitations of the previous approaches you can allow null values and then analyse each parameter in a method body:

```
void foo(String a, Integer b, Integer c) {
    b = b != null ? b : 0;
    c = c != null ? c : 0;
    //...
}

foo("a", null, 2);
```

Now all arguments values must be provided, but the default ones may be null.

4. **Optional class.** This approach is similar to nulls, but uses Java 8 Optional class for parameters that have a default value:

```
void foo(String a, Optional<Integer> bOpt) {
    Integer b = bOpt.isPresent() ? bOpt.get() : 0;
    //...
}

foo("a", Optional.of(2));
foo("a", Optional.<Integer>absent());
```

Optional makes a method contract explicit for a caller, however, one may find such signature too verbose.

5. **Builder pattern.** The builder pattern is used for constructors and is implemented by introducing a separate Builder class:

```
class Foo {
    private final String a;
    private final Integer b;

    Foo(String a, Integer b) {
      this.a = a;
      this.b = b;
    }

    //...
}

class FooBuilder {
  private String a = "";
  private Integer b = 0;

  FooBuilder setA(String a) {
    this.a = a;
    return this;
  }

  FooBuilder setB(Integer b) {
```

```
    Foo build() {
      return new Foo(a, b);
    }
  }

  Foo foo = new FooBuilder().setA("a").build();
```

6. **Maps.** When the number of parameters is too large and for most of them default values are usually used, you can pass method arguments as a map of their names/values:

```
void foo(Map<String, Object> parameters) {
    String a = "";
    Integer b = 0;
    if (parameters.containsKey("a")) {
        if (!(parameters.get("a") instanceof Integer)) {
            throw new IllegalArgumentException("...");
        }
        a = (String)parameters.get("a");
    } else if (parameters.containsKey("b")) {
        //...
    }
    //...
}

foo(ImmutableMap.<String, Object>of(
    "a", "a",
    "b", 2,
    "d", "value"));
```

Please note that you can combine any of these approaches to achieve a desirable result.

Share Improve this answer Follow edited Jun 14 '18 at 23:10

answered Nov 1 '13 at 2:03

Vitalii Fedorenko
99.8k 26 145 111

1 Good explanation. I've never seen return values used like this. For 5) what do the `return this` do? Also, doesn't `FooBuilder().setA("a").build();` since (by definition) the constructor is called first and `FooBuilder()` returns a value, doesn't this mean `.setA("a"):` doesn't get the chance to be called? – Celeritas Jul 18 '14 at 22:37

6 @Celeritas `return this` returns the same object on which the method was called (in the example, `FooBuilder`). This allows chaining of methods in one statement acting on the same object: `new FooBuilder().setA(..).setB(..).setC(..)` etc as opposed to calling each method in a separate statement. – ADTC Aug 27 '14 at 8:44

3 @Celeritas `new FooBuilder()` returns a `FooBuilder` object on which the `setA` method is called. As `setB` is not called, `this.b` retains the default value. Finally the `build` method is called on this `FooBuilder` object. The `build` method creates and returns a `Foo` object which is set to the variable `Foo foo`. Notice that the `FooBuilder` object is not stored in any variable. – ADTC Aug 27 '14 at 8:48

Annotation can also be used to create default parameters and is most useful when required for polymorphic collections. docs.oracle.com/javase/tutorial/java/annotations/declaring.html – Martin Spamer Jan 27 '15 at 9:47

1 Over 900 upvotes on the same answer across two questions. I'm impressed: stackoverflow.com/questions/965690/java-optional-parameters/... – AdamMc331 Nov 28 '15 at 19:19

291

Share Improve this answer Follow

answered Jun 15 '09 at 18:05

Rob H

13.2k 8 38 45

35 Is it so sad? Doing so would introduce potentially ambiguous function signatures. – Trey Jun 15 '09 at 19:46

72 @Trey: languages with default parameters often ditch function overloading since it is then less compelling. So no ambiguity. Beside, Scala added the feature in 2.8, and somehow solved the ambiguity issue (since they kept the overloading for compatibility reasons). – PhiLho May 24 '11 at 13:09

38 I fail to see how parameter defaults prevent function overloading. C# for instance allows overrides and also allows for default initializes. Seems like arbitrary choice, not restriction is the reason. – FlavorScape Nov 30 '13 at 17:43

64 Yeah, lets trade off making the compiler do some extra work and instead make us all write 100000 overloads to give our library users convenience. Good idea. – user562566 May 13 '14 at 8:02

35 @user562566: Whenever I work on a Java project, I get the impression that Java devs are paid/measured by how many lines of code they produce per day – Mark K Cowan Oct 16 '16 at 20:38

87

Unfortunately, yes.

```
void MyParameterizedFunction(String param1, int param2, bool param3=false) {}
```

could be written in Java 1.5 as:

```
void MyParameterizedFunction(String param1, int param2, Boolean... params) {
    assert params.length <= 1;
    bool param3 = params.length > 0 ? params[0].booleanValue() : false;
}
```

But whether or not you should depend on how you feel about the compiler generating a

```
new Boolean[]{}
```

for each call.

For multiple defaultable parameters:

```
void MyParameterizedFunction(String param1, int param2, bool param3=false, int
param4=42) {}
```

could be written in Java 1.5 as:

```
    assert l < 1 || Boolean.class.isInstance(p[0]);
    assert l < 2 || Integer.class.isInstance(p[1]);
    bool param3 = l > 0 && p[0] != null ? ((Boolean)p[0]).booleanValue() : false;
    int param4 = l > 1 && p[1] != null ? ((Integer)p[1]).intValue() : 42;
}
```

This matches C++ syntax, which only allows defaulted parameters at the end of the parameter list.

Beyond syntax, there is a difference where this has run time type checking for passed defaultable parameters and C++ type checks them during compile.

Share Improve this answer Follow edited Oct 3 '13 at 21:37

answered May 26 '10 at 21:51

ebelisle

1,117 7 9

17 Clever, but varargs (...) can only be used for the final parameter, which is more limiting than what languages supporting default parameters give you. – CurtainDog May 26 '10 at 22:20

6 that's clever but a bit messy compared to the C++ version – Someone Somewhere Nov 4 '11 at 18:27

5 Java definitely needs optional defaulted parameters as C# and others allow... the syntax is obvious and I presume they can implement this fairly simply even by just compiling all possible combinations... I cannot imagine why they haven't added it to the language yet! – jwl Feb 23 '12 at 23:31

10 One should never use an `assert` in production code. Throw an exception. – Michael Dorst Oct 1 '13 at 18:52

5 -1 This really isn't what varargs is for. This is a hack. -- in this case, using overloads will be way more readable (which is unfortunate, since three extra characters is more readable than 5 extra lines of source...). -- but Java does not support default parameters. – BrainSlugs83 Apr 18 '15 at 18:43

42

No, but you can very easily emulate them. What in C++ was:

```
public: void myFunction(int a, int b=5, string c="test") { ... }
```

In Java, it will be an overloaded function:

```
public void myFunction(int a, int b, string c) { ... }

public void myFunction(int a, int b) {
    myFunction(a, b, "test");
}

public void myFunction(int a) {
    myFunction(a, 5);
}
```

Earlier was mentioned, that default parameters caused ambiguous cases in function overloading. That is simply not true, we can see in the case of the C++: yes, maybe it can

Share Improve this answer Follow edited Oct 18 '17 at 3:39 answered Nov 4 '14 at 9:12

peterh
1

11 The main idea of C# default value notation is precisely to avoid this boilerplate coding and have just one constructor instead of many. – Kolya Ivankov Oct 17 '17 at 9:27

1 @Kolyalvankov I don't know C#, but I know C++ where the reasoning is the same. I don't know what is the better, but I think, actually the same boilerplate code is generated by the compiler in the case of C++/C# and it is going into the final binary. – peterh Oct 17 '17 at 9:34

8 Every programming language is (in particular) a means to avoid an Assembler boilerplate, am I wrong? The question is only whether it gives a handy functionality, or not. – Kolya Ivankov Oct 17 '17 at 11:45

2 First sentence is a rhetorical question. The word "question" in the second sentence has nothing to do with the rhetorical question in the first. – Kolya Ivankov Oct 17 '17 at 13:44

2 Being more specific: languages are tools that allow us to write programs in a way we can have control over what is written, compiling is a way to tell a machine what we want from it. A tool is more useful if it allows us to avoid boilerplate. In effect, gnavi asked, if they can avoid precisely the type of boilerplate code you propose as an answer, since C# allows for it. – Kolya Ivankov Oct 17 '17 at 13:50

26

You can do this is in Scala, which runs on the JVM and is compatible with Java programs.
http://www.scala-lang.org/

i.e.

```
class Foo(var prime: Boolean = false, val rib: String)  {}
```

Share Improve this answer Follow edited Apr 3 '12 at 13:58 answered Jul 21 '11 at 16:26

om-nom-nom
60.7k 11 177 225

lythic
470 4 7

62 Bring whole new language to get one not so common feature? – om-nom-nom Apr 3 '12 at 13:59

8 @om-nom-nom: Java should never existed. Saying that a feature is not used is equivalent that nobody needs it is saying that Java was not popular before it was invented means that Gosling should not start designing it. – Val Jun 15 '12 at 15:17

29 @Val just saying that this is like shooting birds with cannons – om-nom-nom Jun 15 '12 at 15:22

8 that has nothing to do with the OP's question – destan Nov 7 '16 at 15:45

3 Works in Kotlin, too. And Groovy. And C#. And Javascript. And almost all other languages which are made for real people and problems. – spyro Apr 29 '20 at 15:42

20

Instead of using:

```
void parameterizedMethod(String param1, int param2) {
```

```
    //use all three parameters here
}
```

You could utilize java's Optional functionality by having a single method:

```
void parameterizedMethod(String param1, int param2, @Nullable Boolean param3) {
    param3 = Optional.ofNullable(param3).orElse(false);
    //use all three parameters here
}
```

The main difference is that you have to use wrapper classes instead of primitive Java types to allow `null` input. `Boolean` instead of `boolean`, `Integer` instead of `int` and so on.

Share Improve this answer Follow

answered Feb 14 '20 at 11:52

Viktor Taranenko
219 2 3

i tried but still need to pass params3 into function for avoid warning error from android studio – famfamfam Sep 14 at 4:55

19

No, but **the simplest way to implement this** is:

```
public myParameterizedFunction(String param1, int param2, Boolean param3) {

    param3 = param3 == null ? false : param3;
}

public myParameterizedFunction(String param1, int param2) {

    this(param1, param2, false);
}
```

or instead of the *ternary operator*, you can use `if`:

```
public myParameterizedFunction(String param1, int param2, Boolean param3) {

    if (param3 == null) {
        param3 = false;
    }
}

public myParameterizedFunction(String param1, int param2) {

    this(param1, param2, false);
}
```

Share Improve this answer Follow edited Jun 27 '19 at 11:01

answered Nov 22 '17 at 18:13

simhumileco
23.7k 14 114 98

the modern era, and keeps on designing Java as if it were the 1990s. :D – shevy Mar 5 '20 at 14:34

16

I might be stating the obvious here but why not simply implement the "default" parameter yourself?

```
public class Foo() {
        public void func(String s){
                func(s, true);
        }
        public void func(String s, boolean b){
                //your code here
        }
}
```

for the default, you would either use

```
func("my string");
```

and if you wouldn't like to use the default, you would use

```
func("my string", false);
```

Share Improve this answer Follow

edited Oct 24 '19 at 20:41 OmG 16.5k 7 47 76

answered Feb 18 '13 at 13:59 IronWolf 179 1 5

16 The poster asked if this (rather ugly) pattern can be avoided ... ;-) In more modern languages (like c#, Scala) you don't need this extra overloads which only creates more lines of code. Up to some point you can use varargs in the meantime (static int max(int... array) { }), but they are only a very ugly workaround. – Offler Feb 18 '13 at 14:10

2 Overloading is not ugly and has many benefits, such as different method calls with different signatures can perform different functionality. `//This is better public class Foo() { /* This does something */ public void func(String s){ //do something } /* This does something else with b */ public void func(String s, boolean b){ // b was passed } } //Than this public class Foo() { /* This does something unless b = value, then it does something else */ public void func(String s, boolean b = value){ If (b){ // Do Something } else{ // Do something else } } }` – Antony Booth Nov 14 '13 at 19:57

1 Well, if one wants differing behaviour. If the only difference is a slight change in calculations, etc., it is sure a waste of effort to create multiple signatures. Defaults make sense where you need them to... and lack of it shouldn't be classified as a "useless" requirement. – Kapil Jan 20 '14 at 4:47

@Offler default parameters have nothing to do with "modern language". I used them in Delphi 20 years ago and they probably already existed in Turbo Pascal. – The incredible Jan Apr 6 '17 at 13:09

1 I agree with Offler, and disagree with Antony Booth. I find it not only ugly but also fairly inefficient. Languages such as ruby or python make it trivial to use default parameters; I guess what Java requires you to do is to find (and use) workarounds. The explicit check versus null seems to be the least ugly choice, since I may call that from the commandline too (just not provide anything, and then handle the

8

As Scala was mentioned, Kotlin is also worth mentioning. In Kotlin function parameters can have default values as well and they can even refer to other parameters:

```
fun read(b: Array<Byte>, off: Int = 0, len: Int = b.size) {
    ...
}
```

Like Scala, Kotlin runs on the JVM and can be easily integrated into existing Java projects.

Share Improve this answer Follow

answered Nov 26 '17 at 19:51

mrts
11.9k 5 68 59

6

No. In general Java doesn't have much (any) syntactic sugar, since they tried to make a simple language.

Share Improve this answer Follow

answered Jun 15 '09 at 18:49

tomjen
3,635 3 25 35

29 Not quite. The bitter truth is that the team was on a tight schedule and had no time for syntactic sugar. Why else would `const` and `goto` be reserved keywords which no implementation? — Especially `const` is something I miss bitterly — `final` is no replacement and they knew it. — And if you made the *concious* decision to never implement `goto` you won't need to reserve the keyword. — And later in the Java Team cheated by making the Label based `break` and `continue` as powerful as a Pascal `goto` . – Martin Sep 18 '11 at 12:46

"Simple, Object-Oriented and Familiar" was indeed a design goal - see oracle.com/technetwork/java/intro-141325.html – mikera Sep 2 '13 at 9:27

1 tomjen said: "No. In general Java doesn't have much (any) syntactic sugar, since they tried to make a simple language". So you are saying that removing a lot of unnecessary features from C++ makes a Java a simple language, then tell me why Java does have variadic methods? Why does it have varargs? It is not needed if you can simply use an array of objects instead, am I right? So varargs can be removed from the language, because it is unnecessary. This will make Java more simple than what it is now. Am I right? Overloading also can be removed, because you have infinite names for each method. – user2133061 Aug 8 '17 at 15:58

1 and that's why we got spring and stuff that take the simple language and turn any real Java project into a clusterfuck of syntax and boilerplate :-) – matanster Aug 17 '19 at 8:54

1 Forcing devs to boilerplate code and complicated workarounds is not my understanding of the word "easy". myFunction(a, b=false, c = 3), that's what I call easy. – spyro Apr 29 '20 at 15:44

No.

6 You can achieve the same behavior by passing an Object which has smart defaults. But again it

Share Improve this answer Follow

edited Sep 13 '13 at 17:13

Peter Mortensen
28.9k 21 96 123

answered Jun 15 '09 at 18:16

Santosh Gokak
3,213 3 19 23

4

It is not supported but there are several options like using parameter object pattern with some syntax sugar:

```
public class Foo() {
    private static class ParameterObject {
        int param1 = 1;
        String param2 = "";
    }

    public static void main(String[] args) {
        new Foo().myMethod(new ParameterObject() {{ param1 = 10; param2 = "bar";}});
    }

    private void myMethod(ParameterObject po) {
    }
}
```

In this sample we construct `ParameterObject` with default values and override them in class instance initialization section `{ param1 = 10; param2 = "bar";}`

Share Improve this answer Follow

edited Dec 13 '12 at 17:15

answered Dec 13 '12 at 17:09

hoaz
9,423 4 39 52

3

Try this solution:

```
public int getScore(int score, Integer... bonus)
{
    if(bonus.length > 0)
    {
        return score + bonus[0];
    }

    return score;
}
```

Share Improve this answer Follow

edited Sep 3 '14 at 16:45

answered Nov 3 '13 at 8:36

Hamzeh Soboh
7,150 5 38 54

3

You may use Java Method Invocation Builder to automatically generate the builder with default values.

Just add @GenerateMethodInvocationBuilder to the class, or interface, and the @Default to parameters in methods where you want default values. A builder will be generated at compile

```
@GenerateMethodInvocationBuilder
public class CarService {
 public CarService() {
 }

 public String getCarsByFilter(//
   @Default("Color.BLUE") Color color, //
   @Default("new ProductionYear(2001)") ProductionYear productionYear,//
   @Default("Tomas") String owner//
 ) {
  return "Filtering... " + color + productionYear + owner;
 }
}
```

And then you can invoke the methods.

```
CarService instance = new CarService();
String carsByFilter = CarServiceGetCarsByFilterBuilder.getCarsByFilter()//
  .invoke(instance);
```

Or set any of the default values to something else.

```
CarService instance = new CarService();
String carsByFilter = CarServiceGetCarsByFilterBuilder.getCarsByFilter()//
  .withColor(Color.YELLOW)//
  .invoke(instance);
```

Share Improve this answer Follow edited Jun 13 '16 at 17:42

answered Jun 11 '16 at 18:28

Tomas Bjerre
2,711 19 22

It is not supported in java as in other language for ex. Kotlin.

3

Share Improve this answer Follow

answered Dec 15 '19 at 17:11

Virendra Pal Singh
211 2 11

A similar approach to https://stackoverflow.com/a/13864910/2323964 that works in Java 8 is to use an interface with default getters. This will be more whitespace verbose, but is mockable, and it's great for when you have a bunch of instances where you actually want to draw attention to the parameters.

2

```
public class Foo() {
    public interface Parameters {
        String getRequired();
        default int getOptionalInt(){ return 23; }
        default String getOptionalString(){ return "Skidoo"; }
    }
```

```
    }

    public static void baz() {
        final Foo foo = new Foo(new Person() {
            @Override public String getRequired(){ return "blahblahblah"; }
            @Override public int getOptionalInt(){ return 43; }
        });
    }
}
```

Share Improve this answer Follow

edited May 23 '17 at 11:47
Community Bot
1 1

answered Jul 14 '16 at 17:58
Novaterata
3,540 24 41

2

I've now spent quite some time to figure out how to use this with methods that return values, and I haven't seen any examples so far, I thought it might be useful to add this here:

```
int foo(int a) {
    // do something with a
    return a;
}

int foo() {
    return foo(0); // here, 0 is a default value for a
}
```

Share Improve this answer Follow

answered Mar 12 '19 at 12:17
AAGD
1,297 1 11 17

1

This is how I did it ... it's not as convenient perhaps as having an 'optional argument' against your defined parameter, but it gets the job done:

```
public void postUserMessage(String s,boolean wipeClean)
{
    if(wipeClean)
    {
        userInformation.setText(s + "\n");
    }
    else
    {
        postUserMessage(s);
    }
}

public void postUserMessage(String s)
{
    userInformation.appendText(s + "\n");
}
```

Notice I can invoke the same method name with either just a string or I can invoke it with a string and a boolean value. In this case, setting wipeClean to true will replace all of the text in

Also notice I am not repeating code in the two methods, I am merely adding the functionality of being able to reset the TextArea by creating a new method with the same name only with the added boolean.

I actually think this is a little cleaner than if Java provided an 'optional argument' for our parameters since we would need to then code for default values etc. In this example, I don't need to worry about any of that. Yes, I have added yet another method to my class, but it's easier to read in the long run in my humble opinion.

Share Improve this answer Follow

answered Jan 18 '15 at 15:40

Michael Sims
1,777 13 20

1

NO, But we have alternative in the form of function overloading.

called when no parameter passed

```
void operation(){

int a = 0;
int b = 0;

}
```

called when "a" parameter was passed

```
void operation(int a){

int b = 0;
//code

}
```

called when parameter b passed

```
void operation(int a , int b){
//code
}
```

Share Improve this answer Follow

answered Sep 1 '16 at 6:51

Umair Khalid
2,041 1 18 26

1

There are half a dozen or better issues such as this, eventually, you arrive at the static factory pattern ... see the crypto API for that. Sort difficult to explain, but think of it this way: If you have a constructor, default or otherwise, the only way to propagate state beyond the curly braces is either to have a Boolean isValid; along with the null as default value x failed

Code Correct be damned, I write thousand line constructors and do what I need. I find using isValid at object construction - in other words, two-line constructors - but for some reason, I am migrating to the static factory pattern. I just seem you can do a lot if you in a method call, there are still sync() issues but defaults can be 'substituted' better (safer)

I think what we need to do here is address the issue of null as default value vis-a-vis something String one=new String(""); as a member variable, then doing a check for null before assigning string passed to the constructor.

Very remarkable the amount of raw, stratospheric computer science done in Java.

C++ and so on has vendor libs, yes. Java can outrun them on large scale servers due to it's a massive toolbox. Study static initializer blocks, stay with us.

Share Improve this answer Follow

edited Oct 24 '19 at 20:44

OmG

16.5k 7 47 76

answered Sep 23 '09 at 0:44

Nicholas Jordan

628 3 6

1

One idea is to use `String... args`

```
public class Sample {
   void demoMethod(String... args) {
      for (String arg : args) {
         System.out.println(arg);
      }
   }
   public static void main(String args[] ) {
      new Sample().demoMethod("ram", "rahim", "robert");
      new Sample().demoMethod("krishna", "kasyap");
      new Sample().demoMethod();
   }
}
```

Output

```
ram
rahim
robert
krishna
kasyap
```

from https://www.tutorialspoint.com/Does-Java-support-default-parameter-values-for-a-method

Share Improve this answer Follow

answered Jul 25 '20 at 8:17

Inês Gomes

2,573 1 17 29

```
public MyParameterizedFunction(String param1, int param2, boolean param3)
{
    if(param3 == null) {
        param3 = false;
    }
}
```

However i heavily recommend using something else, like overloading or a static factory. Maybe you can get away with this, but it can lead to unexcspected behavior. For example you could have an error in your code, so that your boolean never gets a value. In this case you would not get a `NullPointerException` . Instead it will look like it was set to false, which can be very confusing to debug.

Share Improve this answer Follow

edited Dec 27 '20 at 14:09

samgiz
123 5

answered Oct 20 '20 at 10:45

Tobias
303 1 12

0

You can use the following-

```
public void mop(Integer x) {
  // Define default values
        x = x == null ? 200 : x;
}
```

Share Improve this answer Follow

answered Dec 30 '19 at 7:50

Sobhit Sharma
415 6 28

0

constructor like for method

```
static void popuping() {
    popuping("message", "title");
}
static void popuping(String message) {
    popuping(message, "title");
}
static void popuping(String message, String title){
    JOptionPane.showMessageDialog(null, message,
            title, JOptionPane.INFORMATION_MESSAGE);
}
```

Share Improve this answer Follow

answered Oct 5 at 1:05

Marco Polo
41 4

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.