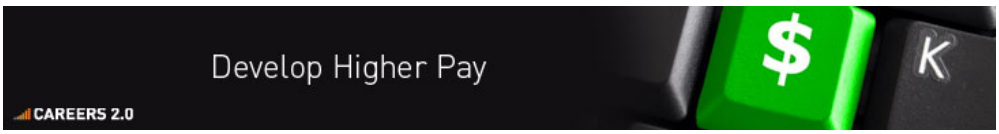
why f is placed after float values?



I don't know why f or F is placed after float values in Java or other languages? for instance,

```
float fVariable = 12.3f;
```

any features other than indicating that this is a float value?

java floating-point

edited Mar 18 '13 at 9:38
Martin Thompson
10.2k 1 10 26

asked Mar 17 '12 at 7:26
ipkiss
2,480 7 30 61

Same Question: Java: Why do you need to specify a "f" in a float literal? – Grijesh Chauhan Jul 6 '13 at 16:39

add comment

4 Answers

by default `12.3` is double literal so to tell compiler to treat as `float` it uses 'f`

answered Mar 17 '12 at 7:28
Jigar Joshi
95.7k 12 125 192

add comment

Seeing as there are only so many ways to represent a number in your program, the designers of Java had to cherry pick and assign each form to the most common use case. For those forms selected as default, the *suffix* that denotes the exact type is optional.

- For Integer literals (int, long), the default is *int*. For obvious reasons.
- For Floating point literals (float, double) the default is *double*. Because using double potentially allows *safer* arithmetic on the stored values.

So, when you type `12` in your program, thats an *int* literal, as opposed to `12L`, which is a *long*. And when you type `12.3`, thats a *double* literal, as opposed to `12.3F`, which is a *float*.

So where is this relevant? Primarily in handling downcasts, or narrowing conversions. Whenever you downcast a long to an int, or a double to a float, the possibility for data loss exists. So, the compiler will force you to indicate that you ***really*** want to perform the narrowing conversion, by signaling a compile error for something like this:

```
float f = 12.3;
```

Because 12.3 represents a double, you have to explicitly cast it to a float (basically signing off on the narrowing conversion). Otherwise, you could indicate that the number is really a float, by using the correct suffix;

```
float f = 12.3f;
```

So too summarize, having to specify a suffix for *longs* and *floats* is a compromise the language designers chose, in order to balance the need to specify what exactly a number is, with the flexibility of converting numbers from one storage type to another.

answered Mar 17 '12 at 10:13

Perception
37.7k 7 34 66

It's worth noting that if `d` is the best `double` representation of some numerical quantity `x`, `float f=(float)d` will be the closest `float` to some value that's within a part per trillion of `x`, even though it requires a typecast. By contrast, if `f` is the best `float` representation of some numerical value `x`, `double d=f` won't require a typecast even though it will often be nowhere close to the best `double` representation of `x`, and may be off by *hundreds of orders of magnitude*. I find it ironic that `float f=(float)(1.0/10.0)` needs a cast, but `double d=1.0f/10.0f` doesn't. – supercat Jun 10 '13 at 17:33

add comment

`float` and `double` can only provide approximate representation values for some values. e.g. 12.3 or 0.1

The difference is that float is not as accurate (as it has less precision, because its smaller)

e.g.

```
System.out.println("0.1f == 0.1 is " + (0.1f == 0.1));
System.out.println("0.1f is actually " + new BigDecimal(0.1f));
System.out.println("0.1 is actually " + new BigDecimal(0.1));
```

prints

```
0.1f == 0.1 is false
0.1f is actually 0.100000001490116119384765625
0.1 is actually 0.1000000000000000055511151231257827021181583404541015625
```

So `0.1` is the closest representation in `double` and `0.1f` is the closest representation in `float`

answered Mar 17 '12 at 8:47

Peter Lawrey
203k 21 162 355

add comment

---

float fVariable = 12.3; is fine. but when you use only float value(without any identifier) in any expression that time you need to tell compiler that value is float hence we use suffix "f" after value. example

float fl =13f/5f;

here 13 and 5 are float values.

answered Mar 17 '12 at 9:20

kundan bora
1,198 3 12

add comment

Not the answer you're looking for? Browse other questions tagged java floating-point or ask your own question.