What are all the different ways to create an object in Java?

Ask Question

Had a conversation with a coworker the other day about this.

There's the obvious which is to use a constructor, but what other ways are there?

java

asked Sep 18 '08 at 18:35
Mike Deck
11.5k 14 55 85

1 When in doubt, look at the language spec. 12.5 Creation of New Class Instances java.sun.com/docs/books/jls/third_edition/html/… 15.9 Class Instance Creation Expressions java.sun.com/docs/books/jls/third_edition/html/… – Internet Friend Sep 18 '08 at 18:42

9 there are 3 only: normal c-tor (new keyword), clone() and `Unsafe.allocateInstance(Class)` . The rest call one of those. Reflection is compiled to c-tor call, deserialization to Unsafe.allocateInstance(Class). You can create your own API and you will end up calling one of those. – bestsss Feb 25 '11 at 18:06

2 @bestsss- `Unsafe` is an implementation-specific detail of Java and isn't mentioned anywhere in the spec. It is entirely possible to build a compliant Java implementation that does not use compile reflection down to code that uses `new` , `clone` , or `Unsafe.allocateInstance` . – templatetypedef Jul 14 '11 at 0:49

2 you could check the link, codesandlogics.com/2017/01/ways-to-create-objects-in-java.html – Pragya Jan 8 '17 at 7:26

javabench.in/2015/04/how-many-ways-we-can-create-object-in.html – Raúl Feb 3 '17 at 14:30

22 Answers

There are four different ways to create objects in java:

A. Using `new` keyword
This is the most common way to create an object in java. Almost 99% of objects are created in this way.

```
MyObject object = new MyObject();
```

B. Using `Class.forName()`
If we know the name of the class & if it has a public default constructor we can create an object in this way.

```
MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();
```

C. Using `clone()`
The clone() can be used to create a copy of an existing object.

```
MyObject anotherObject = new MyObject();
MyObject object = (MyObject) anotherObject.clone();
```

D. Using `object deserialization`
Object deserialization is nothing but creating an object from its serialized form.

```
ObjectInputStream inStream = new ObjectInputStream(anInputStream );
MyObject object = (MyObject) inStream.readObject();
```

You can read from here

edited Apr 28 '15 at 17:55
Anil Chahal
1,437 12 17

answered Feb 24 '11 at 12:26
kamaci
30k 57 187 304

9 So actually only 2 ways exist: calling constructor (using new, clone() or reflection) and deserialization that does not invoke constructor. – AlexR Feb 24 '11 at 12:32

9 @AlexR: `Object.clone()` doesn't invoke constructor too. – axtavt Feb 24 '11 at 12:39

It depends on the clone() implementation. You are right. – AlexR Feb 24 '11 at 12:47

1 As this seems to be the answer at top, could you add the creations of arrays as sub-cases to A and B? (See my answer for details). – Paŭlo Ebermann Feb 25 '11 at 17:09

2 You should also mention the `Constructor` class, which generalizes `Class.newInstance`. – templatetypedef Jul 14 '11 at 0:43

There are various ways:

- Through `Class.newInstance`.
- Through `Constructor.newInstance`.
- Through deserialisation (uses the no-args constructor of the most derived non-serialisable base class).
- Through `Object.clone` (**does not call a constructor**).
- Through JNI (should call a constructor).
- Through any other method that calls a `new` for you.
- I guess you could describe class loading as creating new objects (such as interned `String`s).
- A literal array as part of the initialisation in a declaration (no constructor for arrays).
- The array in a "varargs" (`...`) method call (no constructor for arrays).
- Non-compile time constant string concatenation (happens to produce at least four objects, on a typical implementation).
- Causing an exception to be created and thrown by the runtime. For instance `throw null;` or `"".toCharArray()[0]`.
- Oh, and boxing of primitives (unless cached), of course.
- JDK8 should have lambdas (essentially concise anonymous inner classes), which are implicitly converted to objects.
- For completeness (and Paŭlo Ebermann), there's some syntax with the `new` keyword as well.

edited Feb 25 '11 at 18:18

answered Jan 20 '10 at 17:44

Tom Hawtin - tackline
122k 25 177 264

3 You should add the "normal way", too :-) – Paŭlo Ebermann Feb 25 '11 at 17:10

@Paŭlo Ebermann That's so old school and uncool. (I assumed that what the question meant by "use a constructor (although most, but not all, of the above do use the/a constructor somewhere along the line).) – Tom Hawtin - tackline

Feb 25 '11 at 17:15

actually there only 3 real ways to do it, for which I added comment – bestsss Feb 25 '11 at 18:07

1 can you add some illustration code or helpful link as well – Hussain Akhtar Wahid 'Ghouri' Jun 28 '13 at 18:44

2 You missed one: `java.misc.Unsafe.allocateInstance()` . Though that is nasty for a number of reasons. And actually, deserialization doesn't use the no-args constructor. Under the hood it uses `allocateInstance` or equivalent black magic. – Stephen C Jun 26 '15 at 11:44

Within the Java language, the only way to create an object is by calling its constructor, be it explicitly or implicitly. Using reflection results in a call to the constructor method, deserialization uses reflection to call the constructor, factory methods wrap the call to the constructor to abstract the actual construction and cloning is similarly a wrapped constructor call.

edited Jul 15 '15 at 7:21

answered Sep 18 '08 at 19:01

Confusion

9,173 6 33 64

1 Incorrect. Deserializatio does not call a class's constructor either explicitly or implicitly. – EJP Jul 9 '15 at 10:14

2 I should not have written 'its constructor' and 'the constructor', but rather 'an constructor' and 'a constructor'. In the case of deserialization, the first applicable no-arg constructor is always called. – Confusion Jul 15 '15 at 7:23

1 The default clone implementation does not call any constructor. – Didier L Nov 16 '16 at 16:51

if this is my clone method implementation "return super.clone();". Then it will not invoke constructor. – Mateen May 26 at 3:58

Yes, you can create objects using reflection. For example, `String.class.newInstance()` will give you a new empty String object.

answered Jan 20 '10 at 16:40

Thomas Lötzer

15.1k 15 58 52

1 if i use this its asking me to enclose in a try/catch block. – GuruKulki Jan 20 '10 at 16:43

2 Yes, there are many cases where exceptions can be thrown. See the JavaDoc for newInstance() for examples of what might go wrong. – Thomas Lötzer Jan 20 '10 at 16:55

Cloning and deserialization.

answered Sep 18 '08 at 18:35
Bill the Lizard
276k 152 487 783

Also you can use

```
Object myObj = Class.forName("your.cClass").newInstance();
```

answered Jan 20 '10 at 16:41
Vincent Ramdhanie
87.2k 20 120 168

This should be noticed if you are new to java, every object has inherited from Object

protected native Object clone() throws CloneNotSupportedException;

answered Jan 20 '10 at 16:44
stacker
52.6k 19 109 186

@stacker: Could you please explain how is this related to creating a new object? Thanks. – ryanprayogo Jan 20 '10 at 17:01

4 @ryanprayogo clone() will return a new object (even though the object is a clone of the object that clone() was called on) and is actually the only way to create a new object without the constructor being called. – Thomas Lötzer Jan 20 '10 at 17:04

Also, you can **de-serialize** data into an object. This doesn't go through the class Constructor !

UPDATED : Thanks Tom for pointing that out in your comment ! And Michael also experimented.

> It goes through the constructor of the most derived non-serializable superclass.
> And when that class has no no-args constructor, a InvalidClassException is thrown upon de-serialization.

Please see Tom's answer for a complete treatment of all cases ;-)
is there any other way of creating an object without using "new" keyword in java

edited May 23 '17 at 12:10
Community ♦
1 1

answered Jan 20 '10 at 17:03
KLE
19.1k 2 44 56

1 It does go through a constructor (the no-arg constructor of the most derived non-serialisable superclass). – Tom Hawtin - tackline Jan 20 '10 at 17:42

1 @Tom Oh wow - I did not know that and experimented a bit. Apparently when the most derived non-serializable superclass *does not have* a no-args constructor, it results in an InvalidClassException *being serialized into the stream and thrown upon deserialization!!* - How bizarre is that? – Michael Borgwardt Jan 20 '10 at 22:23

Other ways if we are being exhaustive.

- On the Oracle JVM is Unsafe.allocateInstance() which creates an instance without calling a constructor.
- Using byte code manipulation you can add code to `anewarray` , `multianewarray` , `newarray` or `new` . These can be added using libraries such as ASM or BCEL. A version of bcel is shipped with Oracle's Java. Again this doesn't call a constructor, but you can call a constructor as a seperate call.

answered Feb 24 '11 at 14:10
Peter Lawrey
421k 54 522 891

There is a type of object, which can't be constructed by normal instance creation mechanisms (calling constructors): **Arrays**. Arrays are created with

```
A[] array = new A[len];
```

or

```
A[] array = new A[] { value0, value1, value2 };
```

As Sean said in a comment, this is syntactically similar to a constructor call and internally it is not much more than allocation and zero-initializing (or initializing with explicit content, in the second case) a memory block, with some header to indicate the type and the length.

When passing arguments to a varargs-method, an array is there created (and filled) implicitly, too.

A fourth way would be

```
A[] array = (A[]) Array.newInstance(A.class, len);
```

Of course, cloning and deserializing works here, too.

There are many methods in the Standard API which create arrays, but they all in fact are using one (or more) of these ways.

edited Mar 4 '12 at 22:45

answered Feb 24 '11 at 12:52
Paŭlo Ebermann
57.9k 12 116 176

Granted, you can't define Array constructors, but apart from that the mechanism is the same `new` keyword. Array.newInstance is the only New mechanism here – Sean Patrick Floyd Mar 4 '12 at 18:37

@Sean: It's the same keyword, but it is a quite different internal mechanism, I dare to say. – Paŭlo Ebermann Mar 4 '12 at 20:05

That's true, of course. But on the other hand the different versions of array creation are internally pretty much the same. Just realized your answer was from 2011. Sorry for stirring up old stuff :-) – Sean Patrick Floyd Mar 4 '12 at 21:02

@Sean: No problem, I used this occasion to do some grammar fix. – Paŭlo Ebermann Mar 4 '12 at 22:04

nice job, nobody discussed here about arrays! – Mateen May 26 at 4:03

There are five different ways to create an object in Java,

1. Using `new` keyword → constructor get called

```
Employee emp1 = new Employee();
```

2. Using `newInstance()` method of `Class` → constructor get called

```
Employee emp2 = (Employee)
Class.forName("org.programming.mitra.exercises.Employee")
                                .newInstance();
```

It can also be written as

```
Employee emp2 = Employee.class.newInstance();
```

3. Using `newInstance()` method of `Constructor` → constructor get called

```
Constructor<Employee> constructor = Employee.class.getConstructor();
Employee emp3 = constructor.newInstance();
```

4. Using `clone()` method → no constructor call

```
Employee emp4 = (Employee) emp3.clone();
```

5. Using deserialization → no constructor call

```
ObjectInputStream in = new ObjectInputStream(new FileInputStream("data.obj"));
Employee emp5 = (Employee) in.readObject();
```

First three methods `new` keyword and both `newInstance()` include a constructor call but later two clone and deserialization methods create objects without calling the constructor.

All above methods have different bytecode associated with them, Read Different ways to create objects in Java with Example for examples and more detailed description e.g. bytecode conversion of all these methods.

However one can argue that creating an array or string object is also a way of creating the object but these things are more specific to some classes only and handled directly by JVM, while we can create object of any class by using these 5 ways.

edited Aug 8 '17 at 13:08

answered Nov 19 '16 at 5:55
Naresh Joshi
1,259 11 18

Reflection:

```
someClass.newInstance();
```

answered Sep 18 '08 at 18:37
John Meagher
11.5k 11 44 53

Reflection will also do the job for you.

```
SomeClass anObj = SomeClass.class.newInstance();
```

is another way to create a new instance of a class. In this case, you will also need to handle the exceptions that might get thrown.

answered Jan 20 '10 at 16:44
ryanprayogo
8,110 9 41 63

- using the `new` operator (thus invoking a constructor)
- using reflection `clazz.newInstance()` (which again invokes the constructor). Or by `clazz.getConstructor(..).newInstance(..)` (again using a constructor, but you can thus choose which one)

To summarize the answer - one main way - by invoking the constructor of the object's class.

Update: Another answer listed two ways that do not involve using a constructor - deseralization and cloning.

edited Feb 24 '11 at 12:28

answered Feb 24 '11 at 12:23

Bozho
463k 102 913 1037

You can also clone existing object (if it implements Cloneable).

```
Foo fooClone = fooOriginal.clone ();
```

answered Jan 20 '10 at 17:08

Roman
34.5k 71 201 289

There are FIVE different ways to create objects in Java:

1. Using `new` keyword:

This is the most common way to create an object in Java. Almost 99% of objects are created in this way.

```
MyObject object = new MyObject();//normal way
```

2. By Using Factory Method:

```
ClassName ObgRef=ClassName.FactoryMethod();
```

Example:

```
RunTime rt=Runtime.getRunTime();//Static Factory Method
```

3. By Using Cloning Concept:

By using `clone()` , the `clone()` can be used to create a copy of an existing object.

```
MyObjectName anotherObject = new MyObjectName();
MyObjectName object = anotherObjectName.clone();//cloning Object
```

4. Using `Class.forName()`:

If we know the name of the class & if it has a public default constructor we can create an object in this way.

```
MyObjectName object = (MyObjectNmae)
Class.forName("PackageName.ClassName").newInstance();
```

Example:

```
String st=(String)Class.forName("java.lang.String").newInstance();
```

5. Using object deserialization:

Object deserialization is nothing but creating an object from its serialized form.

```
ObjectInputStreamName inStream = new ObjectInputStreamName(anInputStream );
MyObjectName object = (MyObjectNmae) inStream.readObject();
```

edited May 22 '12 at 9:58
ServAce85
921 1 17 42

answered May 14 '11 at 7:35
K.V.Subrahmanya Reddy
47 1

(4) only requires `Class.forName()` if you don't already have the class, which in all the other cases you do. It also doesn't require a no-args constructor: there are ways to call any public constructor if you know the correct arguments. And you've left out at least two other ways. – EJP May 22 '12 at 8:15

2 (2) Factory Method is just a pattern for getting objects. But internally it uses "new" keyword for creating objects. – Karthik Bose Dec 17 '13 at 7:59

man why are so many people saying factory method create objects, where did you guys learn this from ? – Mateen May 26 at 4:02

From an API user perspective, another alternative to constructors are static factory methods (like BigInteger.valueOf()), though for the API author (and technically "for real") the objects are still created using a constructor.

answered Sep 18 '08 at 20:10
Fabian Steeg
37.9k 5 71 107

Method 1

Using new keyword. This is the most common way to create an object in java. Almost 99% of objects are created in this way.

```
Employee object = new Employee();
```

Method 2

Using Class.forName(). Class.forName() gives you the class object, which is useful for reflection. The methods that this object has are defined by Java, not by the programmer writing the class. They are the same for every class. Calling newInstance() on that gives you an instance of that class (i.e. callingClass.forName("ExampleClass").newInstance() it is equivalent to calling new ExampleClass()), on which you can call the methods that the class defines, access the visible fields etc.

```
Employee object2 = (Employee) Class.forName(NewEmployee).newInstance();
```

Class.forName() will always use the ClassLoader of the caller, whereas ClassLoader.loadClass() can specify a different ClassLoader. I believe that Class.forName initializes the loaded class as well, whereas the ClassLoader.loadClass() approach doesn't do that right away (it's not initialized until it's used for the first time).

Another must read:

Java: Thread State Introduction with Example Simple Java Enum Example

Method 3

Using clone(). The clone() can be used to create a copy of an existing object.

```
Employee secondObject = new Employee();
Employee object3 = (Employee) secondObject.clone();
```

Method 4

Using newInstance() method

```
Object object4 =
Employee.class.getClassLoader().loadClass(NewEmployee).newInstance();
```

Method 5

Using Object Deserialization. Object Deserialization is nothing but creating an object from its serialized form.

```
// Create Object5
// create a new file with an ObjectOutputStream
FileOutputStream out = new FileOutputStream("");
ObjectOutputStream oout = new ObjectOutputStream(out);

// write something in the file
oout.writeObject(object3);
oout.flush();

// create an ObjectInputStream for the file we created before
ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("crunchify.txt"));
Employee object5 = (Employee) ois.readObject();
```

edited Jul 9 '15 at 9:49
EJP
251k 22 197 335

answered Jul 9 '15 at 9:28
Andriya
133 8

Don't use code formatting for text that isn't code. There are more methods than these. Read the other answers. 'Almost 99%' is just guesswork. – EJP Jul 9 '15 at 9:50

Hi EJP sorry for this mistake...I said this is one kind of way to create the objects not said exactly its the right one.Its just a model..and sorry am alearner and new one to stackoverflow – Andriya Jul 9 '15 at 10:05

check youtu.be/gGGCmrD6Qpw to see complete code example – nanosoft Aug 29 '17 at 18:31

Depends exactly what you mean by create but some other ones are:

- Clone method
- Deserialization
- Reflection (Class.newInstance())
- Reflection (Constructor object)

answered Sep 18 '08 at 18:40

Garth Gilmour
8,753 3 17 29

2 3 and 4 are different aliases for the Same mechanism – Sean Patrick Floyd Mar 4 '12 at 18:31

there is also ClassLoader.loadClass(string) but this is not often used.

and if you want to be a total lawyer about it, arrays are *technically* objects because of an array's .length property. so initializing an array creates an object.

answered Sep 18 '08 at 20:20
the0ther
5,911 8 33 63

1 loadClass(String name) returns resulting Class object which an object yes But not that class's object. If examples are given then its we can find numerous such examples throughout java library but those will be class specific. check youtu.be/gGGCmrD6Qpw – nanosoft Aug 29 '17 at 18:30

We can create an objects in 5 ways:

1. by new operator
2. by reflection (e.g. Class.forName() followed by Class.newInstance())
3. by factory method
4. by cloning
5. by reflection api

edited Nov 22 '17 at 22:48
Garth Gilmour
8,753 3 17 29

answered Aug 26 '11 at 13:26
tanushree roy
19 1

3 Class.forName() loads a class rather than creating an object. – EJP May 22 '12 at 8:15

reflexion? Surely you mean reflection. – Stephen C Jun 26 '15 at 11:36

how would you create object from factory method, internal implementation may be again using new key word right? and why do you have reflection twice ? It would make more sense if you actually give some exampls – Mateen May 26 at 3:53

We can also create the object in this way:-

```
String s ="Hello";
```

Nobody has discuss it.

answered Jun 24 '14 at 6:39

Deepak Sharma
3,074 3 37 48

This is the way of creating primitive data types, it is just a flexibility that Java provides behind the scenes to not use the "new" keyword.This is the same as the new keyword. – Madusudanan Jun 24 '14 at 7:04

Madhusudan, FYI , With the help of new operator, objects should always store in heap while in this case "Hello" is an object which should store in String pool. And String is a class not a primitive datatype . – Deepak Sharma Jun 24 '14 at 13:32

This doesn't create an object. It assigns a reference to an existing object. The object had already been created by the compiler and classloader. – EJP Jul 9 '15 at 9:59
