



[Course](#) [Progress](#)

[Course](#) > [Readings/Videos](#) > [Reading 12: Equality](#) > [Questions](#)

[<](#) Previous

























Next [>](#)

## Questions

 [Bookmark this page](#)

### autoboxing confusion

3/3 points (graded)

Here's the code again from the end of the previous section:

```
Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();  
a.put("c", 130); // put ints into the map  
b.put("c", 130);
```

What is the compile-time type of the expression `130` ?



After executing `a.put("c", 130)`, what is the runtime type that is used to represent the value 130 in the map?



What is the compile-time type of `a.get("c")` ?



[Submit](#)

 [Show Answer](#)

 Correct (3/3 points)

### autoboxing confusion, part 2

2/2 points (graded)

```
Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();  
a.put("c", 130); // put ints into the map  
b.put("c", 130);
```


Draw a snapshot diagram after the code above has executed. How many HashMap objects are in your snapshot diagram?

 Answer: 2

#### Explanation

The code creates two new `HashMap` objects, one assigned to `a` and the other assigned to `b`.

How many Integer objects are in your snapshot diagram?

 Answer: 2

#### Explanation

Putting 130 into each HashMap creates a fresh Integer object for each one.

Submit

Show Answer

Answers are displayed within the problem

### autoboxing confusion, part 3

2/2 points (graded)

```
Map<String, Integer> a = new HashMap<>(), b = new HashMap<>();  
a.put("c", 130); // put ints into the map  
b.put("c", 130);
```

After this code executes, what would `a.get("c").equals(b.get("c"))` return?

true ☐ ☒ Answer: true

#### Explanation

Both `get()` calls return an `Integer` object representing 130. Since `equals()` is correctly implemented for the (immutable) `Integer` type, it returns true for those two values.

What would `a.get("c") == b.get("c")` return?

false ☐ ☒ Answer: false

#### Explanation

The `get()` calls return *distinct* `Integer` objects, so they are not referentially equal. `==` returns false.

This is the surprising pitfall: if you have in your mind that the `Map` contains `int` values, you will be surprised by the behavior of `get()`, because it returns an `Integer` instead. Most of the time you can use `Integer` interchangeably with `int`, but not when it comes to equality operators like `==` and `equals`. You can see [this code in action](#) on Online Java Tutor.

Submit

Show Answer

Answers are displayed within the problem

### autoboxing confusion, part 4

1/1 point (graded)

Now suppose you assign the `get()` results to `int` variables:

```
int i = a.get("c");  
int j = b.get("c");  
boolean isEqual = (i == j);
```

After executing this code, what is the value of `isEqual`?

true ☐ ☒ Answer: true

#### Explanation

The assignments automatically *unbox* the `Integer` objects into `int` values, both 130. Those primitive `int` values are both 130, so `==` now returns true. You can see [this code in action](#) on Online Java Tutor.

Submit

Show Answer

Answers are displayed within the problem

< Previous

Next >

[Open Learning Library](#)

[About](#)

[Accessibility](#)

[All Courses](#)

[Why Support MIT Open Learning?](#)

[Help](#)

[Connect](#)

[Contact](#)

[Twitter](#)

[Facebook](#)

[Privacy Policy](#) [Terms of Service](#)

© Massachusetts Institute of Technology, 2024