Explain different Integer Cache (-128..127), Float, and double caches

**ChatGPT said:**

Absolutely! Let’s break down **caching of wrapper objects** in Java for Integer, Long, Short, Byte, Character, Boolean, Float, and Double, and explain **how it works and why**.

**1️⃣ Integer Cache**

* **Class:** java.lang.Integer
* **Method:** Integer.valueOf(int i)
* **Default cache range:** **-128 to 127**
* **Reason:** These values are **used frequently** in programs (loop counters, small constants).

**Behavior:**

Integer a = Integer.valueOf(100);

Integer b = Integer.valueOf(100);

System.out.println(a == b); // true → same cached object

Integer x = Integer.valueOf(200);

Integer y = Integer.valueOf(200);

System.out.println(x == y); // false → different objects

* Cache size can be increased using **JVM option**:

-XX:AutoBoxCacheMax=<size>

* **HashMap impact:** works fine since equals() compares values.

**2️⃣ Long, Short, Byte caches**

* **Long:** Long.valueOf(long l) → caches **-128..127**
* **Short:** Short.valueOf(short s) → caches **-128..127**
* **Byte:** Byte.valueOf(byte b) → caches **all byte values (-128..127)**

Example:

Byte b1 = 100;

Byte b2 = 100;

System.out.println(b1 == b2); // true, cached

Short s1 = 127;

Short s2 = 127;

System.out.println(s1 == s2); // true, cached

* Idea: small primitive wrapper values are **frequently used**, caching avoids unnecessary object creation.

**3️⃣ Character cache**

* **Class:** Character.valueOf(char c)
* **Default cache range:** **0 to 127** (ASCII range)
* **Reason:** Most common characters (ASCII) are reused frequently.

Example:

Character c1 = 'A';

Character c2 = 'A';

System.out.println(c1 == c2); // true

* Characters beyond 127 → new objects created.

**4️⃣ Boolean cache**

* **Class:** Boolean.valueOf(boolean b)
* Only **two objects exist:** Boolean.TRUE and Boolean.FALSE
* Always returns **same object** for true/false.

Example:

Boolean t1 = Boolean.valueOf(true);

Boolean t2 = Boolean.valueOf(true);

System.out.println(t1 == t2); // true

**5️⃣ Float and Double cache**

* **Float:** Float.valueOf(float f)
* **Double:** Double.valueOf(double d)
* **Cache range:** Only **-128 to 127**, **step = 1** for integers
* **Decimals are not cached** → only integer-like float/double in the range are cached

Example:

Float f1 = Float.valueOf(100.0f);

Float f2 = Float.valueOf(100.0f);

System.out.println(f1 == f2); // true → cached

Float f3 = Float.valueOf(100.5f);

Float f4 = Float.valueOf(100.5f);

System.out.println(f3 == f4); // false → new objects

* Same behavior applies for Double.valueOf().

**6️⃣ Why caching exists**

* **Reduce memory allocation** for frequently used numbers
* **Improve performance** for boxing/unboxing operations
* **Works seamlessly in collections** (HashMap, HashSet, etc.) because .equals() always compares values, not references.

**7️⃣ Quick reference table**

| **Type** | **Cache Range** | **Notes** |
| --- | --- | --- |
| Integer | -128 to 127 | Configurable via -XX:AutoBoxCacheMax |
| Long | -128 to 127 |  |
| Short | -128 to 127 |  |
| Byte | -128 to 127 (all values) |  |
| Character | 0 to 127 (ASCII) |  |
| Boolean | true/false only | Only two objects exist |
| Float | -128 to 127 (integer only) | Decimals not cached |
| Double | -128 to 127 (integer only) | Decimals not cached |