**Primitive and Object data type**

**1.  Primitive data types trong Java là gì?**

**Primitive data types** là các kiểu dữ liệu đơn giản, không phải là đối tượng, và lưu trữ giá trị trực tiếp. Java cung cấp 8 kiểu dữ liệu nguyên thủy:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Type** | **Description** | **Default** | **Size** | **Example Literals** | **Range of values** |
| **boolean** | true or false | false | 8 bits | true, false | true, false |
| **byte** | twos-complement integer | 0 | 8 bits | (none) | -128 to 127 |
| **char** | Unicode character | \u0000 | 16 bits | ‘a’, ‘\u0041’, ‘\101’, ‘\\’, ‘\’, ‘\n’, ‘β’ | characters representation of ASCII values  0 to 255 |
| **short** | twos-complement integer | 0 | 16 bits | (none) | -32,768 to 32,767 |
| **int** | twos-complement intger | 0 | 32 bits | -2,-1,0,1,2 | -2,147,483,648  to  2,147,483,647 |
| **long** | twos-complement integer | 0 | 64 bits | -2L,-1L,0L,1L,2L | -9,223,372,036,854,775,808  to  9,223,372,036,854,775,807 |
| **float** | IEEE 754 floating point | 0.0 | 32 bits | 1.23e100f , -1.23e-100f , .3f ,3.14F | upto 7 decimal digits |
| **double** | IEEE 754 floating point | 0.0 | 64 bits | 1.23456e300d , -123456e-300d , 1e1d | upto 16 decimal digits |

**2. Object data type trong java là gì?**

**Object data type** trong Java (còn được gọi là **Reference data type**) là loại dữ liệu mà biến không trực tiếp lưu trữ giá trị, mà lưu trữ **tham chiếu** (reference) đến một đối tượng trong bộ nhớ heap. Giá trị mặc định của các biến kiểu object là null nếu chưa được khởi tạo.

**3. Phân biệt kiểu dữ liệu nguyên thủy và kiểu dữ liệu object**

* **Primitive types** lưu trữ giá trị trực tiếp trong bộ nhớ.

o   int a = 10; // giá trị 10 được lưu trực tiếp

* **Object types** (hay còn gọi là Reference types) lưu trữ tham chiếu (địa chỉ) của đối tượng trong bộ nhớ, không phải giá trị trực tiếp.

o   **String str = "**Hello World**"; // địa chỉ tham chiếu của chuỗi "Hello" được lưu trong biến 'str'.**

o   String str = "Hello World";  
System.*out*.println("Địa chỉ của biến str: " + Integer.*toHexString*(System.*identityHashCode*(str)));

o   Địa chỉ của biến str: 6ce253f1

**4. Có thể chuyển đổi giữa hai kiểu dữ liệu này không?**

* **Boxing**: là quá trình tự động chuyển một primitive type thành một object type (wrapper class).
* **Unboxing**: là quá trình chuyển một object type (wrapper class) thành primitive type.

-        boolean -> Boolean

-        byte -> Byte

-        short -> Short

-        int -> Integer

-        long -> Long

-        float -> Float

-        double -> Double

-        char -> Character

**5. Có thể so sánh hai kiểu dữ liệu này với nhau không?**

* Không thể so sánh hai kiểu dữ liệu nguyên thủy và object data type trực tiếp với nhau.
* Nếu muốn so sánh thì phải chuyển các kiểu dữ liệu nguyên thủy thành kiểu wrapper class.
* Khi so sánh **primitive types**, toán tử == được sử dụng để so sánh giá trị trực tiếp
* Khi so sánh **object types**, == so sánh địa chỉ (reference) của đối tượng, còn .equals() so sánh nội dung của đối tượng.

**6. Giá trị khi khởi tạo biến với hai loại kiểu dữ liệu này là gì?**

-        Primitive Data Types

·  boolean: giá trị mặc định là false.

·  byte: giá trị mặc định là 0.

·  short: giá trị mặc định là 0.

·  int: giá trị mặc định là 0.

·  long: giá trị mặc định là 0L.

·  float: giá trị mặc định là 0.0f.

·  double: giá trị mặc định là 0.0d.

·  char: giá trị mặc định là ký tự trống ('\u0000'), hay còn gọi là ký tự null.

-        **Object (Reference) types**: giá trị mặc định là null, tức là chưa tham chiếu đến bất kỳ đối tượng nào.

**7.**     **Tại sao Java sử dụng primitive types và wrapper classes**

1. Lý do sử dụng Primitive Types

Primitive types là các kiểu dữ liệu cơ bản nhất, mang lại hiệu suất và hiệu quả bộ nhớ tốt hơn. Lý do chính Java sử dụng chúng bao gồm:

a. Hiệu suất và Tối ưu bộ nhớ

* **Hiệu suất cao**: Primitive types lưu trữ trực tiếp các giá trị trong bộ nhớ stack (một vùng nhớ nhanh và nhỏ gọn hơn). Vì thế, khi sử dụng các phép toán tính toán trên primitive types như int, double, Java thực hiện nhanh hơn so với các đối tượng.
* **Tiết kiệm bộ nhớ**: Các primitive types sử dụng ít bộ nhớ hơn so với các đối tượng (wrapper classes). Ví dụ, một int chỉ chiếm 4 byte, trong khi Integer có thể chiếm nhiều hơn vì nó là một đối tượng với dữ liệu bổ sung và lưu trữ tham chiếu trong heap memory.

b. Đơn giản và Dễ sử dụng

* Primitive types trực tiếp thể hiện các giá trị đơn giản như số, ký tự và logic (boolean), dễ sử dụng cho các thao tác tính toán hoặc so sánh.
* Với các kiểu dữ liệu như int, float, boolean, không cần phải lo về việc khởi tạo đối tượng hay quản lý bộ nhớ, điều này giúp việc lập trình đơn giản hơn.

2. Lý do sử dụng Wrapper Classes

Wrapper classes là các đối tượng cung cấp nhiều tính năng hơn ngoài **việc chỉ lưu trữ giá tr**ị, và chúng cần thiết trong nhiều trường hợp mà primitive types không thể sử dụng được. Những lý do chính Java sử dụng wrapper classes bao gồm:

a. Tương thích với Collections và Generics

* **Các collection framework (như ArrayList, HashMap) chỉ làm việc với các đối tượng (object types)**, vì vậy không thể lưu trữ trực tiếp các primitive types như int, boolean. Các wrapper classes (như Integer, Boolean) cung cấp cách bọc (wrap) giá trị nguyên thủy thành đối tượng để có thể sử dụng trong các collection.
* **Generics**: Java Generics yêu cầu các đối tượng, vì vậy nếu bạn muốn sử dụng các kiểu dữ liệu nguyên thủy với Generics, bạn cần dùng các wrapper class.

b. Các phương thức hữu ích

* Các wrapper classes cung cấp nhiều phương thức tiện lợi cho việc xử lý dữ liệu. Ví dụ, Integer cung cấp các phương thức như parseInt(), toString(), compareTo(), cho phép dễ dàng chuyển đổi, so sánh, hoặc thực hiện các thao tác trên dữ liệu số nguyên.

c. Khả năng chứa giá trị null

* **Nullability**: Wrapper classes có thể chứa giá trị null, điều mà các primitive types không thể làm được. Điều này hữu ích khi bạn cần biểu diễn trạng thái "không có giá trị" hoặc "giá trị không hợp lệ" trong các hệ thống, đặc biệt khi làm việc với cơ sở dữ liệu hoặc API.

**8. Sự khác biệt giữa == và .equals() khi so sánh dữ liệu kiểu primitive và object là gì?**

1. So sánh bằng ==

* **Đối với kiểu dữ liệu nguyên thủy (primitive)**: Toán tử == so sánh **giá trị** của hai biến. Nếu hai giá trị là giống nhau, kết quả trả về là true, ngược lại là false.
* **Đối với object**: Toán tử == so sánh **địa chỉ tham chiếu** của hai object, tức là nó kiểm tra xem cả hai có trỏ đến cùng một vùng nhớ hay không. Nếu hai object là cùng một tham chiếu, == trả về true, ngược lại là false.

2. So sánh bằng .equals()

* **Đối với kiểu dữ liệu nguyên thủy (primitive)**: Không thể sử dụng .equals() cho kiểu dữ liệu nguyên thủy, vì nó là một phương thức của object. Các kiểu nguyên thủy như int, boolean, char,... không phải là object, nên bạn không thể gọi .equals() trên chúng.
* **Đối với object**: Phương thức .equals() được sử dụng để so sánh **giá trị (nội dung)** của hai object. Nhiều class như String, Integer, và các lớp wrapper khác trong Java đã override phương thức .equals() để so sánh giá trị thay vì so sánh tham chiếu.