1. Immutable and Mutable

* An object is considered *immutable* if its state cannot change after it is constructed. Maximum reliance on immutable objects is widely accepted as a sound strategy for creating simple, reliable code.
* Immutable objects are particularly useful in concurrent applications. Since they cannot change state, they cannot be corrupted by thread interference or observed in an inconsistent state.
* Programmers are often reluctant to employ immutable objects, because they worry about the cost of creating a new object as opposed to updating an object in place. The impact of object creation is often overestimated, and can be offset by some of the efficiencies associated with immutable objects. These include decreased overhead due to garbage collection, and the elimination of code needed to protect mutable objects from corruption.
* The following subsections take a class whose instances are mutable and derives a class with immutable instances from it. In so doing, they give general rules for this kind of conversion and demonstrate some of the advantages of immutable objects.

1. String trong Java
2. String

A screenshot of a computer

Description automatically generated with low confidence

1. String Buffer

A picture containing text, font, line, algebra

Description automatically generated

1. String Builder

A close-up of text

Description automatically generated with low confidence

A picture containing text, font, line, screenshot

Description automatically generated

1. String pool
2. Integer pool trong Java

* Như bạn thấy, phương thức**Integer.valueOf()** trả về một thể hiện Integer đại diện cho giá trị int được chỉ định.
* Nếu một giá trị kiểu int nằm trong khoảng **Integer.low** và **Integer.high** thì đối tượng Integer được trả về từ **Integer.cache**. Trường hợp ngược lại, khởi tạo đối tượng Integer mới, đại diện cho giá trị int được chỉ định.

A picture containing text, screenshot, font, document

Description automatically generated

* **Integer.cache** lưu trữ bộ nhớ đệm (cache) các giá trị Integer. Như bạn thấy giá, mặc định giá trị **high = 127**, **low = -128**, nên các giá trị được cache trong khoảng **từ -128 đến 127**. Chúng ta có thể thay đổi giá trị **high** bằng cách thêm cấu hình **JVM**:
* -Djava.lang.Integer.IntegerCache.high=<size> hoặc
* -XX:AutoBoxCacheMax=<size>
* Kết quả của ví dụ trên, có thể được giải thích thông qua hình bên dưới:

A picture containing text, screenshot, diagram, font

Description automatically generated

* Khi so sánh bằng toán tử **==** nghĩa là ta so sánh địa chỉ của vùng nhớ. Trường hợp i1 và i2 cùng tham chiếu đến 1 vùng nhớ nên kết quả là true, các trường hợp khác do không cùng vùng nhớ nên kết quả là false. Trường hợp i1 == i7 có kết quả là true, lý do một kiểu Integer Wrapper khi so sánh == với một kiểu int primitive type thì JVM sẽ sử dụng cơ chế [Unboxing](https://gpcoder.com/2066-autoboxing-va-unboxing-trong-java/)để chuyển kiểu Wrapper về kiểu primitive tương ứng.
* Lưu ý: để so sánh giá trị của 2 đối tượng kiểu Integer chúng ta nên sử dụng phương thức **equals()** để so sánh hoặc chuyển đổi về kiểu primitive tương ứng trước khi thực hiện so sánh **==**.

A picture containing text, screenshot, font, number

Description automatically generated