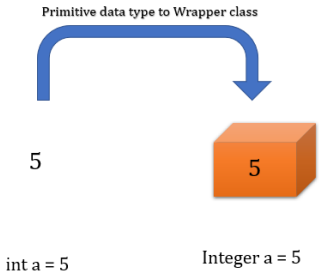
WRAPPER CLASSES

Wrapper Classes-Introduction

A Wrapper class in Java is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store primitive data types.



Need of Wrapper Classes

There are certain needs for using the Wrapper class in Java as mentioned below:

- They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
- The classes in java.util package handles only objects and hence wrapper classes help in this case also.
- Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
- An object is needed to support synchronization in multithreading.

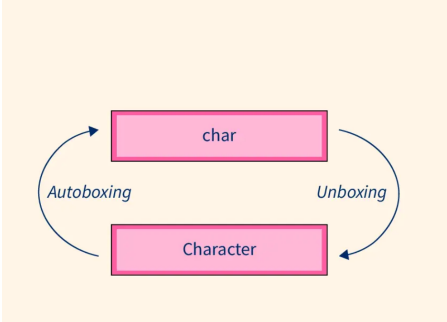
Primitive Data Types and Corresponding Wrapper Class

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean

Autoboxing and Unboxing

Autoboxing The automatic conversion of primitive types to the object of their corresponding wrapper classes is known as autoboxing. For example – conversion of int to Integer, long to Long, double to Double, etc.

Unboxing It is just the reverse process of autoboxing. Automatically converting an object of a wrapper class to its corresponding primitive type is known as unboxing. For example – conversion of Integer to int, Long to long, Double to double, etc.



Custom Wrapper Classes

Java Wrapper classes wrap the primitive data types. We can create a class that wraps data inside it. So let us check how to create our own custom wrapper class in Java. It can be implemented for creating certain structures like queues, stacks, etc.

'final' Keyword

The final method in Java is used as a non-access modifier applicable only to a variable, a method, or a class. It is used to restrict a user in Java.

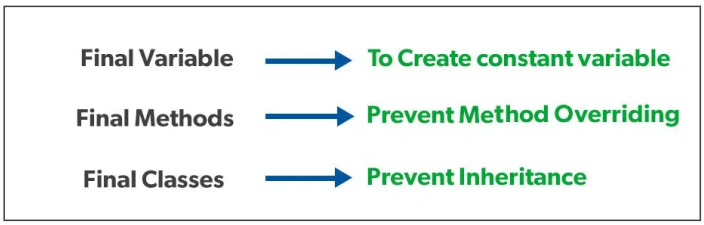
If the final variable is a **reference**, this means that the variable cannot be **re-bound to reference** another object, but the internal state of the object pointed by that reference variable can be changed i.e. you can add or remove elements from the final array or final collection.

The following are **Different Contexts** where the 'final' is used:

- Variable
- Method
- Class

Final Variables: When a variable is declared as final, its value cannot be changed once it has been initialized. This is useful for declaring constants or other values that should not be modified.

Final Methods: When a method is declared as final, it cannot be overridden by a subclass. This is useful for methods that are part of a class's public API and should not be modified by subclasses.



Final Classes: When a class is declared as final, it cannot be extended by a subclass. This is useful for classes that are intended to be used as is and should not be modified or extended.

Initialisation of Variables Using 'final'

When a variable is declared with the ***final keyword***, its **value can't be changed**, essentially, a **constant**. This also means that you **must initialize a final variable**.

The difference between C++ *const* variables and Java *final* variables. const variables in C++ must be **assigned a value when declared**. For final variables in Java, it is **not.** A final variable can be assigned value later, but only once. *final with for-each statement is a legal statement*. Since the "*i*" variable goes out of scope with each iteration of the loop, it is re-declared each iteration, allowing the same token (i) to be used to represent multiple variables.

A final variable is called a **blank final variable** if it is **not** initialized while declaration. Below are the two ways to initialize a blank final variable:

- A blank final variable can be initialized inside an instance-initializer block or the constructor. If you have more than one constructor in your class then it must be initialized in all of them, otherwise, a compile-time error will be thrown.

- A blank final static variable can be initialized inside a static block.

Reference 'final' Variable

When a final variable is a reference to an object, then this final variable is called the **reference final variable**.

As we all know a final variable cannot be re-assign. But in the case of a reference final variable, the internal state of the object pointed by that reference variable can be changed. Note that this is not re-assigning. This property of the *final* is called ***non-transitivity***.