Java: Immutable and mutable strings and primitive objects

Objectives

- Work with Primitive Wrappers,
- immutability, autoboxing, overloading

Table of Content

Immutable Objects and their advantages

- An object is immutable, if the state of the object cannot be changed once it is created.
- String class is *immutable* because any modification methods on string object (like replace() or substring()) does not change the original string.
- Immutable objects are thread-safe!
- Therefore, they are useful in applications that have multiple threads concurrently executing.

Tell me how

- How to create a thread-safe object?

  - The thread safe class should not allow any manipulations to its member variables after creation. What will you do in your class to make this happen?

  - Make member variables **`final`**
  - Provide no setters
  - Don’t allow subclasses to make the object mutable. So either make the class itself as **`final`** or provide a **`private`** constructor.
  - Also if member variables are instances of other classes, they themselves have to be made mutable by this class– it gets more complicated!

Disadvantages of Immutable objects

- Some implementations create and return a new object if immutable objects are changed (like `String` which we have seen). The cost of creating a new object is more compared to updating an object in place.
- Immutable objects are not always desirable for thread-safe code. For instance, does Account object being immutable make sense!

  In cases where the object has to undergo a lot of updations by design, then it is desirable to have mutable objects with synchronized methods.

Wrapper classes

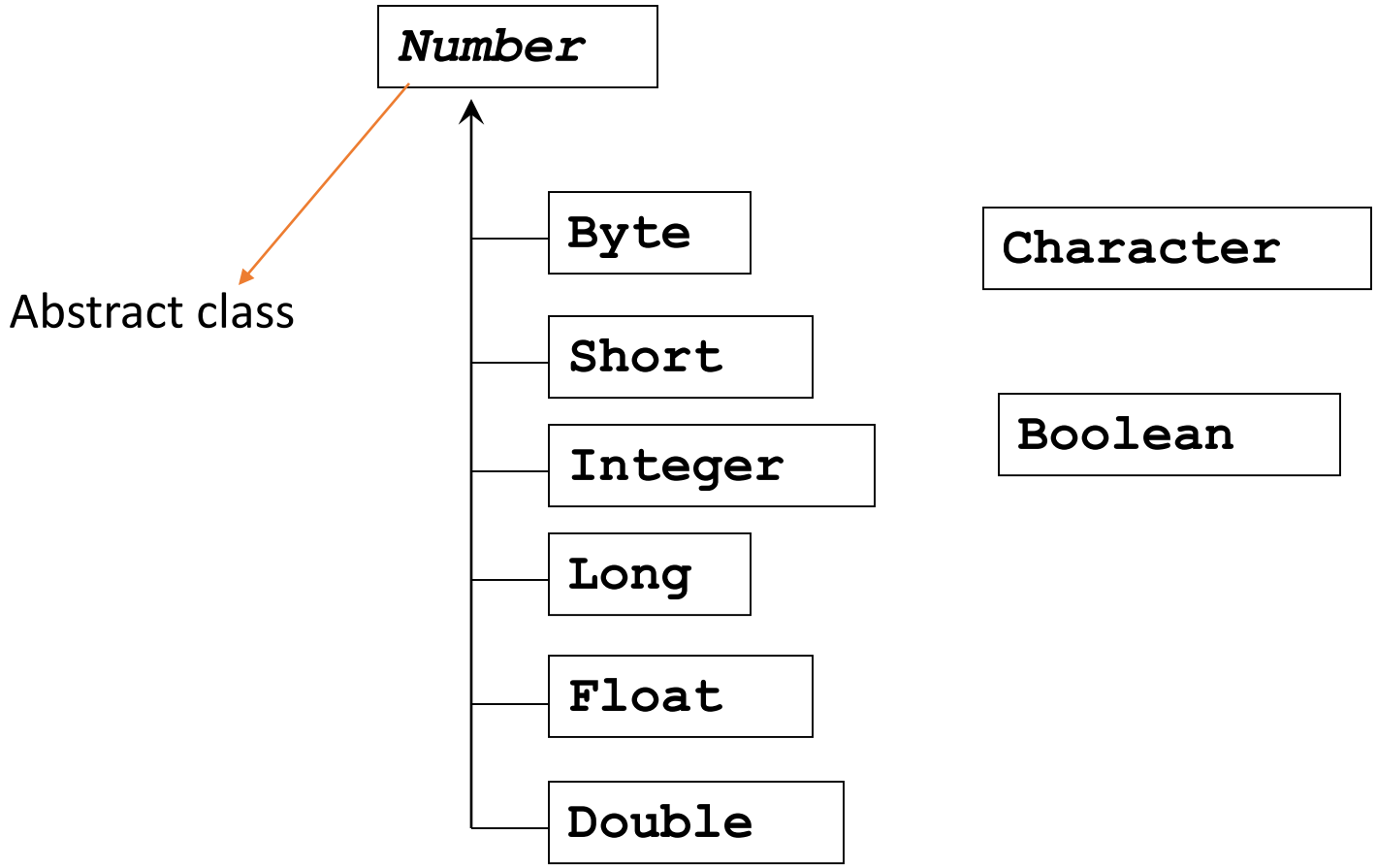
- Wrapper classes in java are classes that wrap other classes.
- Therefore wrapper class' constructor will always take the object of the class they wrap as a parameter.

Did we come across this term before?

The two important packages that have wrapper classes are in **`java.lang`** and **`java.io`**

Primitive Wrappers

- There are some special kinds of wrapper classes in **`java.lang`** package which wrap primitive types.
- All wrapper classes are **`final`** classes except **`Number`** which is **`abstract`** class.



Why should one wrap the primitive types?

1. For various conversions
   - These classes have methods, that allow conversion between string and numeric types

2. Collection works with objects
   - Collection framework in java has numerous classes like **`LinkedList`**, **`ArrayList`** etc that allow storing collection of objects. To add primitives into Collections, need to wrap them into objects before adding

3. Serialization
   - Java stores object state in the hard disk and this is called serialization. But this technique is available only for objects. Therefore if we want to save primitives we need to wrap them inside a class. Instead of this we can use wrapper classes.

Constructors and common methods

- General form of constructor for all wrapper classes:
  a) Constructor with its corresponding primitive type
  b) Constructor with a **`String`** type. Excludes: **`Character`**

  Example:
  `Integer(String)` and **`Integer(int)`**
  `Double(String)` and **`Double(double`**)
- **`Float`** has an extra constructor that takes **`double`**.

  `Float(double)`
- Also, note that

  ~~`Byte b= new Byte(1);`~~ **`// compilation error`**
- Also, all the classes have implemented **`equals()`**, **`toString() and compareTo() methods`**

Exercise

- *Consider the following rules for the evaluation of a physical fitness for a post in a company where only men can apply.*

Age	Weight in pounds	Height in inches
Below 20	Not Eligible	Not Eligible
20 to 30	155 to 175	5’2” to 5’5”
30 to 40	170 to 180	5’4” to 5’6”
40 to 50	175 to 185	5’6” to 6’0”
Above 50	Not Eligible	Not Eligible

Prompt the user for the name, gender, age, weight with units (units could be kgs or lbs, for example , user could enter 90 kgs or 70 lbs) and height with units (units could be in cm or inches, for example , user could enter 5.5 inches or 165 cm).

cont…

If the user enters in pounds and inches, cross check against the table and display whether he is eligible or not. If the user enters in kgs and cms, convert the values and cross check against the table and display whether he is eligible or not. If a woman candidate tries to apply, display a message stating that they are not eligible for the post.

Hint: *1 kg equals 2.2 pounds*
1 Inch = 2.54 cm.

(45 mins)

Exercise

Write a java program which constructs a login name by using the inputs supplied by the user which are Full Name (Ex. R.Anand, Shanti Mohan) and the Account Number (5533881763. The rules are based on the following.

- *First 4 characters of the login name should be the alternate 4 characters of the full name including the initials, starting from the beginning.*
- *Last 4 characters of the login name should be the alternate 4 digits in the account number ,starting from the end.*
- *Whitespace is not allowed.*
- *First letter should be in uppercase and rest all should be in lower case*
- *No special character is allowed.*
- *Hint: Build a character array and then use appropriate String constructor to build the string.*

(45 mins)

Autoboxing

- Autoboxing refers to the automatic conversion of primitive to wrapper ype, also called Boxing
- Boxing and unboxing may hit performance. So be careful where you use it.
- Boxing Examples
  1. `Integer ii=10; ii++;`
  2. `Boolean bb=true; if(bb){}`
  3. `Long ll=34L;` but `Long ll=34` leads to error
  4. `Byte bt=34;` but `Byte bt= 1000;` leads to error
  5. `Float fl= 3.14f;` but `Float fl= 3.14;` leads to error
  6. `Double dl= 3.14;`
  7. `Character c='a';`
- Unboxing is the reverse. A Wrapper object is automatically convertible to its primitive..
  - `int i=ii; boolean b=bb; long l=ll;`

Pros and Cons

- When compiler allows **`Integer ii=10,`** what it does is, it converts the code to

  `Integer ii = new Integer(10);`

- Advantage of boxing are
  - the code is neat and clutter free.
  - Less coding for developers
- Disadvantages of boxing is
  - Experiments and experience yields the fact that when boxing conversions are used within a loop, it affects the performance of a program. Therefore while allowing both wrapper and primitives gives greater flexibility in a collection (like array or List (coming up)), care must be taken when to use them. For instance, if only primitive **`int`** is required, then restricted **`int`** array can be created instead of an **`Integer`** array. Or **`Integer List.`**

- ```
  static void change(int i){
  System.out.println("int");}
  static void change(Integer i){
  System.out.println("Integer");}
  ```
  Call: `change(123);`

What will the code print?
Try out if you are in doubt.

- ```
  static void change(int i){
  System.out.println("int");}
  static void change(Byte i){
  System.out.println(“Byte");}
  ```
  Call: `byte b1=45; change(b1);`

- ```
  static void change(char i){
  System.out.println(“char");}
  static void change(Byte i){
  System.out.println("Byte");}
  ```
  Call: `byte b1=45; change(b1);`
  ```

- ```
  static void change(Integer i){
  System.out.println("integer");}
  ```

  ```
  static void change(Number i){
  System.out.println("number");}
  ```

  Call: `change(45);`

- ```
  static void change(Number i){
  System.out.println(“number");}
  ```

  ```
   static void change(int i){
  System.out.println(“int");}
  ```

  Call: `Byte b1=45;  change(b1);`

- ```
  static void change(Integer i){
  System.out.println("integer");}
  ```

  ```
  static void change(Number i){
  System.out.println("number");}
  ```

  Call: `byte b1=25;change(b1);`
  ```

- ```
  static void change(Object i){
  System.out.println("Object");}

  static void change(Long i){
  System.out.println("Long");}
  ```
  Call: `change(10);`

- ```
  static void change(Integer i){
  System.out.println("integer");}

  static void change(int... i){
  System.out.println("int");}
  ```
  Call: `change(10);`

- ```
  static void change(Integer... i){
  System.out.println("Integer");  }

  static void change(int... i ){
  System.out.println("int");      }
  ```
  Call: `change(10); // error`

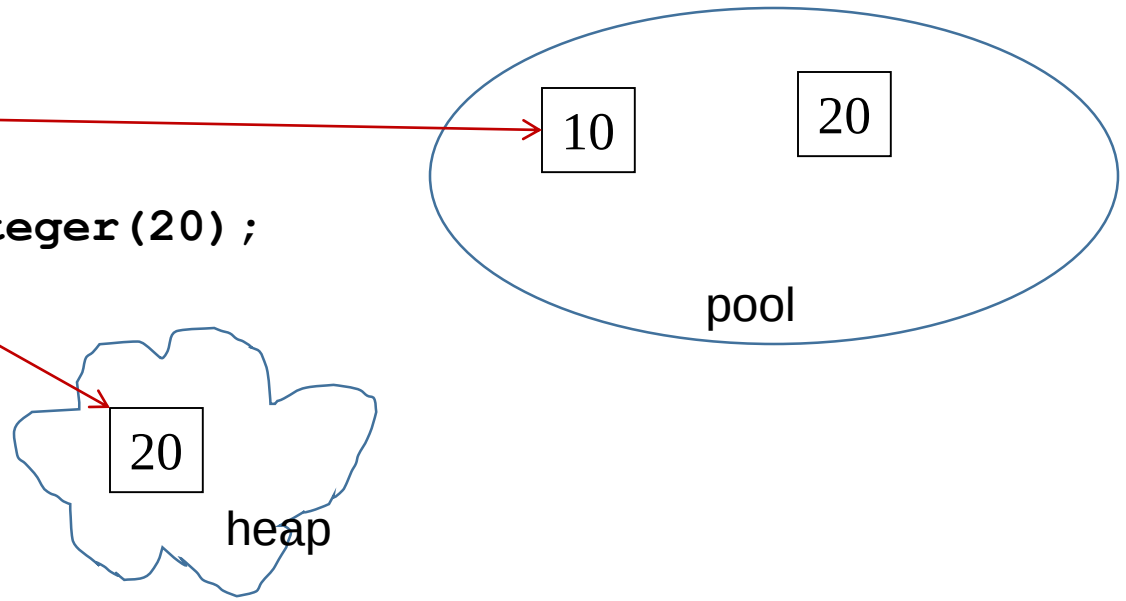
Can you figure out the sequence in which compiler resolves overloaded methods?
  ```

Immutability

- Like **`String`**, Wrapper objects are also immutable.
- So, using 'new' creates primitives in the pool (if it is not present already) as well as in the heap and directly assigning a literal will create primitives in the heap alone.
- That is why there are no setters in wrapper classes.

```
Integer int i=10;

Integer j= new Integer(20);
```



Summary

- String class is immutable because any modification methods on string object does not change the original string. Immutable objects are thread-safe.
- Wrapper classes in java are classes that wrap other classes.
- The two important packages that have wrapper classes are in are java.lang and java.io.
- There are some special kinds of wrapper classes in java.lang package which wrap primitive types.
- Autoboxing refers to the automatic conversion of wrapper class type to its primitive type, which is also called Boxing
- Unboxing is the reverse. A Wrapper object is automatically convertible to its primitive..
- Like String, Wrapper objects are also immutable