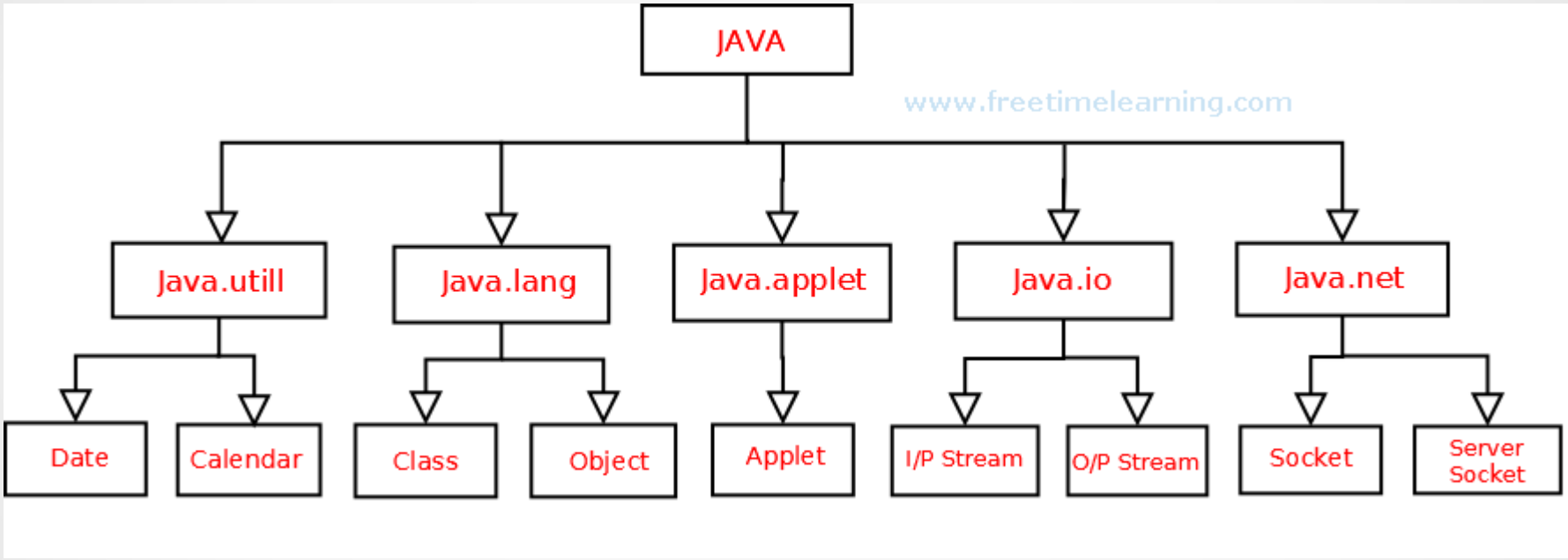
Introduction - PACKAGE

- A java package is a mechanism for organizing Java classes into groups.
- A java package is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two form:
  - built-in package
  - user-defined package.
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
- **Advantage of Java Package**
  - Java package is used to categorize the classes and interfaces so that they can be easily maintained.
  - Java package provides access protection.
  - Java package removes naming collision.

Introduction - PACKAGE



Packages in Java

- A package declaration resides at the top of a Java source file.
- All source files to be placed in a package have common package name.
- A package provides a unique namespace for the classes it contains.
- A package can contain:
  - Classes
  - Interfaces
  - Enumerated types
  - Annotations
- Two classes in different packages can have the same name.
- Packages provide a mechanism to hide its classes from being used by programs or packages belonging to other classes.

Packages in Java

- **How to compile java package**

  javac -d directory javafilename

- For example

  javac -d . Simple.java

- The -d switch specifies the destination where to put the generated class file. You can use any directory name like /home (in case of Linux), d:/abc (in case of windows) etc. If you want to keep the package within the same directory, you can use . (dot).

- **How to run java package program**

  To Compile: javac -d . Simple.java

  To Run: java mypack.Simple

Package1.java

Output:Welcome to package

Packages in Java

How to access package from another package?

There are three ways to access the package from outside the package.

import packagename.*;

import package.classname;

fully qualified name.

Package2.java Package3.java

Package4.java Package5.java

Package6.java Package7.java

1) Using packagename.*

- If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.
- The import keyword is used to make the classes and interface of another package accessible to the current package.

Packages in Java

2) Using packagename.classname

- If you import package.classname then only declared class of this package will be accessible.

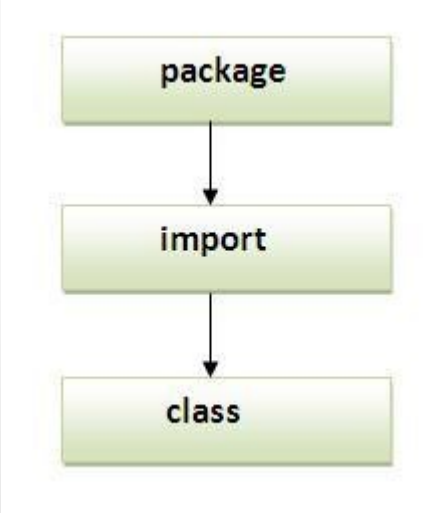
3) Using fully qualified name

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

- It is generally used when two packages have same class name e.g. java.util and java.sql packages contain Date class.

Subpackages in Java

- If you import a package, subpackages will not be imported.
- If you import a package, all the classes and interface of that package will be imported excluding the classes and interfaces of the subpackages.
- Hence, you need to import the subpackage as well.
- Sequence of the program must be package then import then class.



Subpackages in Java

- Package inside the package is called the subpackage. It should be created to categorize the package further.
- The standard of defining package is domain.company.package
- e.g. abc.java.bean

Package8.java Package9.java

Access Modifiers in Java

- There are two types of modifiers in java:
  - access modifiers
  - non-access modifiers
- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
- There are 4 types of java access modifiers:
  - Private
  - Default
  - Protected
  - Public
- There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient etc.

Access Modifiers in Java

Private access modifier: Access1.java Access2.java

- The private access modifier is accessible only within class.
- If you make any class constructor private, you cannot create the instance of that class from outside the class.
- A class cannot be private or protected except nested class.

Access3.java Access4.java

Default access modifier:

- If you don't use any modifier, it is treated as default by default.
- The default modifier is accessible only within package.

Protected access modifier Access5.java Access6.java

- The protected access modifier is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Access Modifiers in Java

Public access modifier

Access8.java

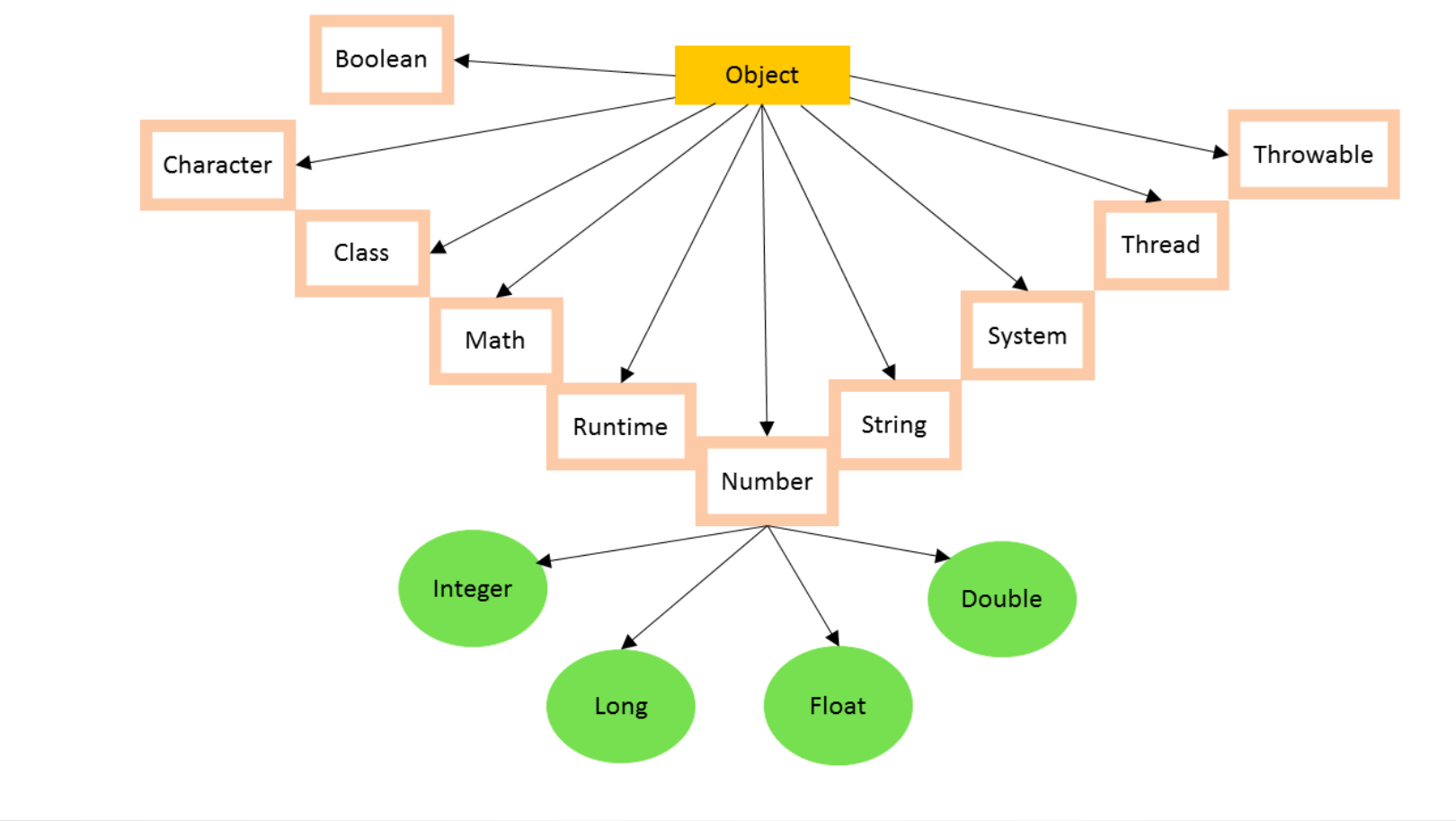
- The public access modifier is accessible everywhere.
- It has the widest scope among all other modifiers.

Access modifier	Within class	Within package	Outside package within subclass	Outside package
private	YES	NO	NO	NO
default	YES	YES	NO	NO
protected	YES	YES	YES	NO
public	YES	YES	YES	YES

Java.lang package

- Java.lang is imported by default in all the classes that we create.
- There is no need to explicitly import the lang package.
- It contains classes that form the basic building blocks of Java.
- There are 37 classes in java.lang package.
- Some of them are as follows:
  - Boolean
  - Byte
  - Double
  - Float
  - Integer
  - Long
  - Object
  - Short

Java.lang.Object class



Java.lang.Object class

- The Object class is the parent class of all the classes in java by default.
- In other words, it is the topmost class of java.
- The Object class is beneficial if you want to refer any object whose type you don't know.
- There is no need to explicitly inherit the Object class.

Java.lang.Object class

Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

Java Wrapper Classes

- Wrapper class in java provides the mechanism to convert primitive into object and object into primitive.
- For each primitive type, there is a corresponding wrapper class designed.
- An instance of wrapper contains or wraps a primitive value of the corresponding type.
- Wrappers allow for situations where primitives cannot be used but their corresponding objects are required.
- The eight classes of java.lang package are known as wrapper classes in java.
- The list of eight wrapper classes are given below:

Java Wrapper Classes

Wrapper Classes for Primitive Data Types	
Primitive Data Types	**Wrapper Classes**
int	Integer
short	Short
long	Long
byte	Byte
float	Float
double	Double
char	Character
boolean	Boolean

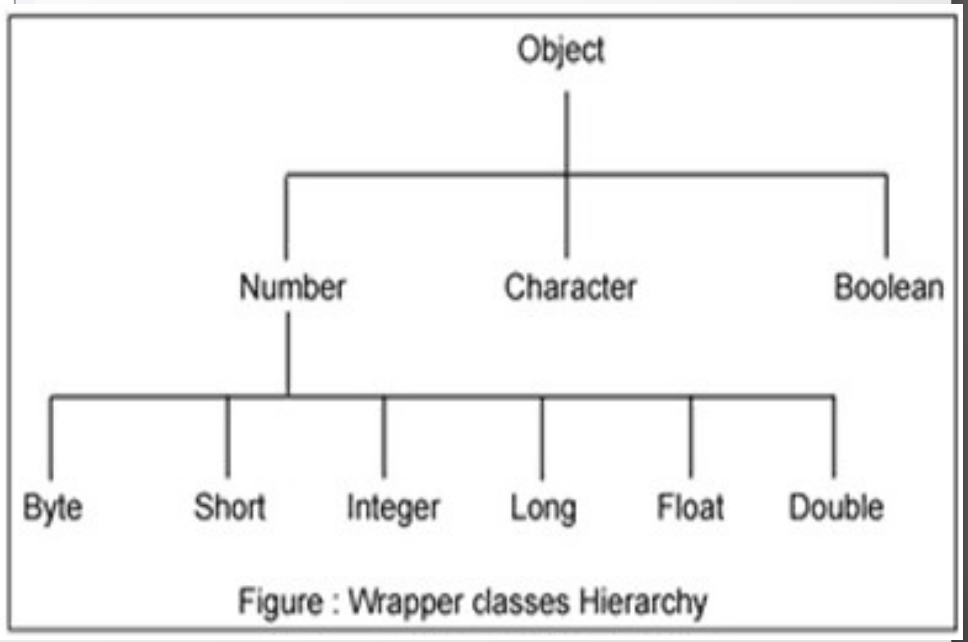


Figure : Wrapper classes Hierarchy

Java Wrapper Classes

- **Features of Wrapper Classes:**
  - All the wrapper classes except Character and Float have constructors. - one that takes the primitive value and another that takes the String representation of the value. Character has one constructor and float has three.
  - Just like strings, wrapper objects are also immutable. i.e once a value is assigned it cannot be changed.

Wrapper classes

- The wrapper classes have a number of static methods for handling and manipulating primitive data types and objects.
- **Constructors:** converting primitive types to wrapper classes

  Integer i = new Integer (10);
- **Methods:** converting wrapper objects to primitives

  all numeric classes have methods to convert a numeric wrapper class to their respective primitive type.

  byteValue(), intValue(), floatValue(), doubleValue(), longValue(), shortValue()

  int v = a.intValue();

Class1.java Class3.java Class4.java

Wrapper classes

- **Converting Primitives to String object:**

  the method toString() is used to convert primitive number data types to String

  String xyz = Integer.toString (v) //converting primitive integer to String

  String xyz = Float.toString (x)

- **Converting Back from String Object to Primitives:**

  the six parser methods are parseInt, parseDouble, parseFloat, parseLong, parseByte and parseShort.

  int v = Integer.parseInt (xyz);

Class5.java

Wrapper classes

- Wrapper classes are mainly used to wrap the primitive content into an object.
- This operation of wrapping primitive content into an object is called **boxing.**
- The reverse process i.e unwrapping the object into corresponding primitive data is called **Unboxing.**
- From JDK 1.5 onwards, **Auto-Boxing** is introduced. According to this feature, you need not to explicitly wrap the primitive content into an object. Just assign primitive data to corresponding wrapper class reference variable, java automatically wraps primitive data into corresponding wrapper object. Class6.java
- From JDK 1.5 onwards, **Auto-Unboxing** is introduced. According to this feature, you need not to call method of wrapper class to unbox the wrapper object. Java implicitly converts wrapper object to corresponding primitive data if you assign wrapper object to primitive type variable. Class7.java Class8.java

String class

- The java.lang.String class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as **trimming, concatenating, converting, comparing, replacing strings etc.**
- Java String is a powerful concept because everything is treated as a string if you submit any form in window based, web based or mobile application.

toUpperCase() and toLowerCase() method

- The java string toUpperCase() method converts this string into uppercase letter and string toLowerCase() method into lowercase letter.

  Syntax:

  String toLowerCase()

  String toUpperCase()

  Example:

  String s="Sachin";

  System.out.println(s.toUpperCase());//SACHIN

  System.out.println(s.toLowerCase());//sachin

  System.out.println(s);//Sachin(no change in original)

String trim() method

- The string trim() method eliminates white spaces before and after string.

Syntax:

String trim()

Example:

```
String s="  Sachin  ";
System.out.println(s);//  Sachin
System.out.println(s.trim());//Sachin
```

String3.java

String startsWith() and endsWith() method

- This method has two variants and tests if a string starts or ends with the specified prefix.

Syntax:

boolean startsWith(String prefix)

boolean endsWith(String prefix)

Example:

```
String s="Sachin";
System.out.println(s.startsWith("Sa"));//true
System.out.println(s.endsWith("n"));//true
```

String1.java

String charAt() method

- The string charAt() method returns a character at specified index.

Syntax:

char charAt(int index)

Example:

```
String s="Sachin";
System.out.println(s.charAt(0));//S
System.out.println(s.charAt(3));//h
```

String length() method

- The string length() method returns length of the string. The length is equal to the number of 16-bit Unicode characters in the string.

  Syntax:

  int length()

  Example:

  String s="Sachin";

  System.out.println(s.length());//6

String valueOf() method

- The string valueOf() method converts given type such as int, long, float, double, boolean, char and char array into string.

Syntax:

String valueOf(double d)

Example:

```
int a=10;
String s=String.valueOf(a);
System.out.println(s+10);  //1010
```

String replace() method

- The string replace() method replaces all occurrence of first sequence of character with second sequence of character.

Syntax:

String replace(char oldChar, char newChar)

Example:

String s1="Java is a programming language. Java is a platform. Java is an Island.";

String replaceString=s1.replace("Java","Kava");//replaces all occurrences of "Java" to "Kava"

String2.java

String substr() method

- This method has two variants and returns a new string that is a substring of this string.
- The substring begins with the character at the specified index and extends to the end of this string or up to endIndex – 1, if the second argument is given.

Syntax

public String substring(int beginIndex, int endIndex)

Example:

```
String Str = new String("Welcome to Java SY A");

System.out.println(Str.substring(10, 13) ); //Java
```

String compareTo() method

- This method compares two strings lexicographically.

Syntax

int compareTo(String anotherString)

The value 0 if the argument is a string equal to this string; a value less than 0 if the argument is a string greater than this string; and a value greater than 0 if the argument is a string less than this string.

```
String str1 = "Strings are immutable";
String str2 = "Strings are immutable";
String str3 = "Integers are not immutable";
  int result = str1.compareTo( str2 ); //0
System.out.println(result);
result = str2.compareTo( str3 ); //10
System.out.println(result);
result = str3.compareTo( str1 ); //-10
System.out.println(result);
```

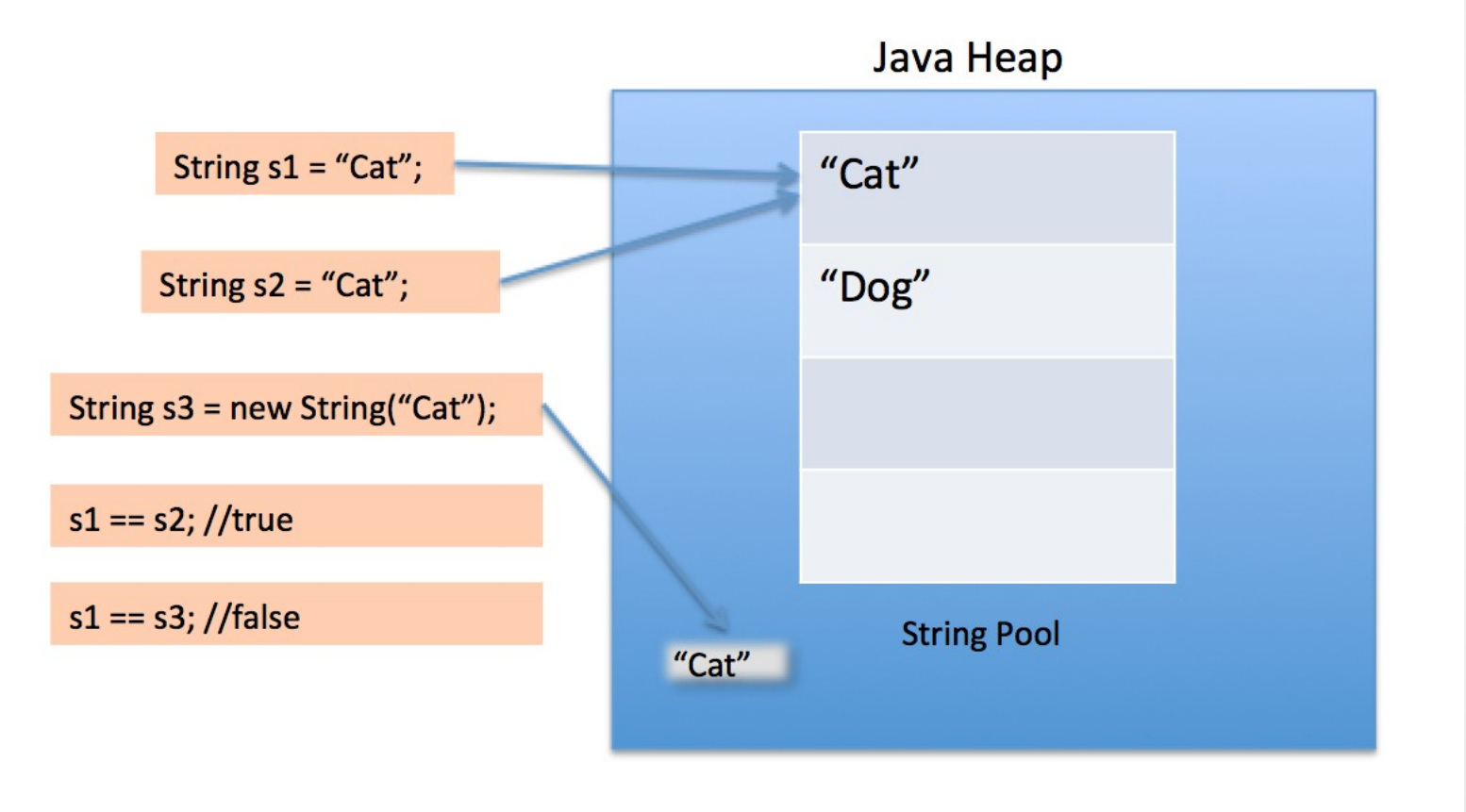
String Buffer class

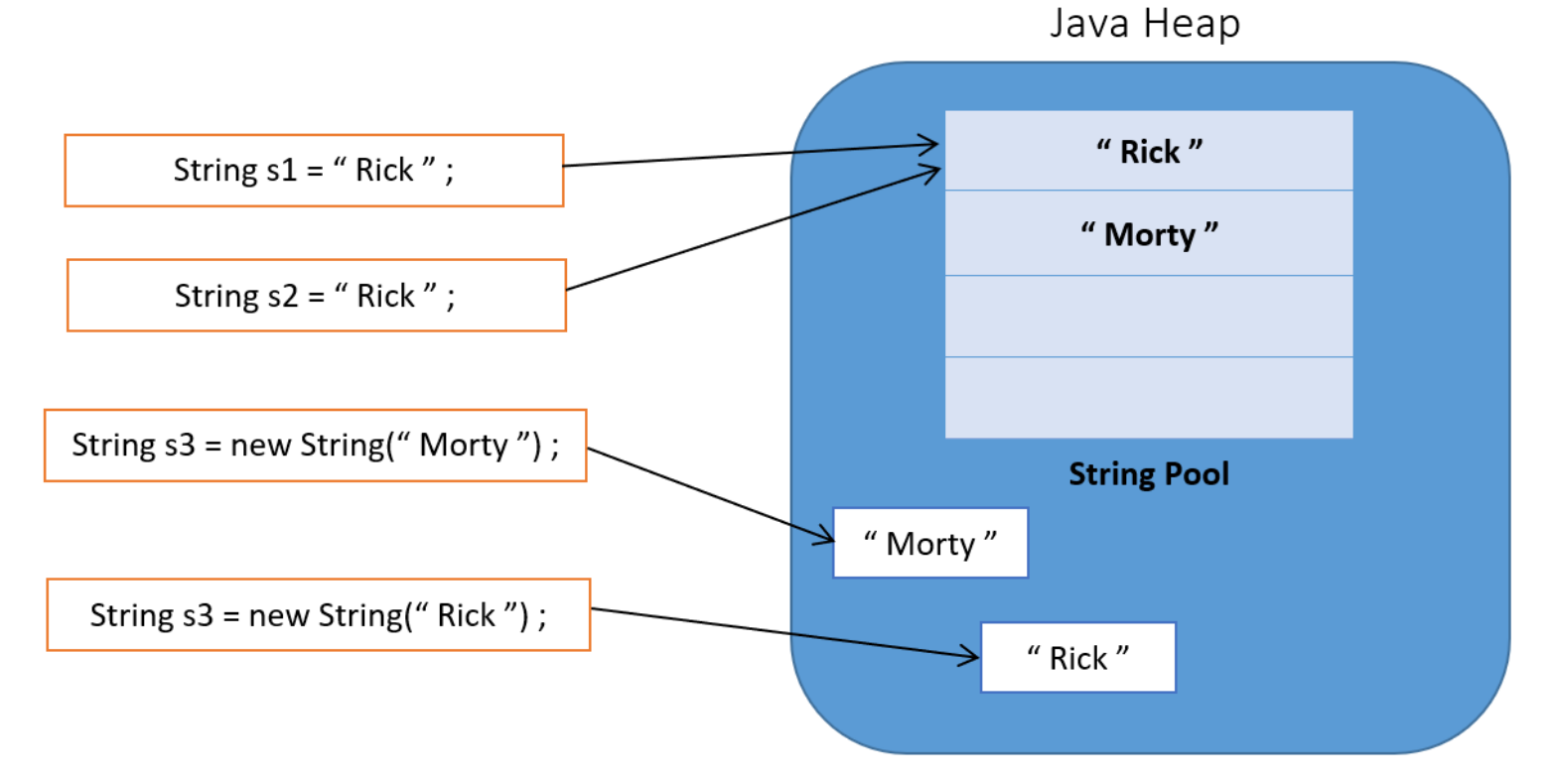
- The java.lang.StringBuffer class is a thread-safe, mutable sequence of characters.

  Following are the important points about StringBuffer:

- A string buffer is like a String, but can be modified.
- It contains some particular sequence of characters, but the length and content of the sequence can be changed through certain method calls.
- They are safe for use by multiple threads.
- Every string buffer has a capacity.

No.	String	StringBuffer
1)	The String class is immutable.	The StringBuffer class is mutable.
2)	String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance.	StringBuffer is fast and consumes less memory when we concatenate t strings.
3)	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.
4)	String class is slower while performing concatenation operation.	StringBuffer class is faster while performing concatenation operation.
5)	String class uses String constant pool.	StringBuffer uses Heap memory

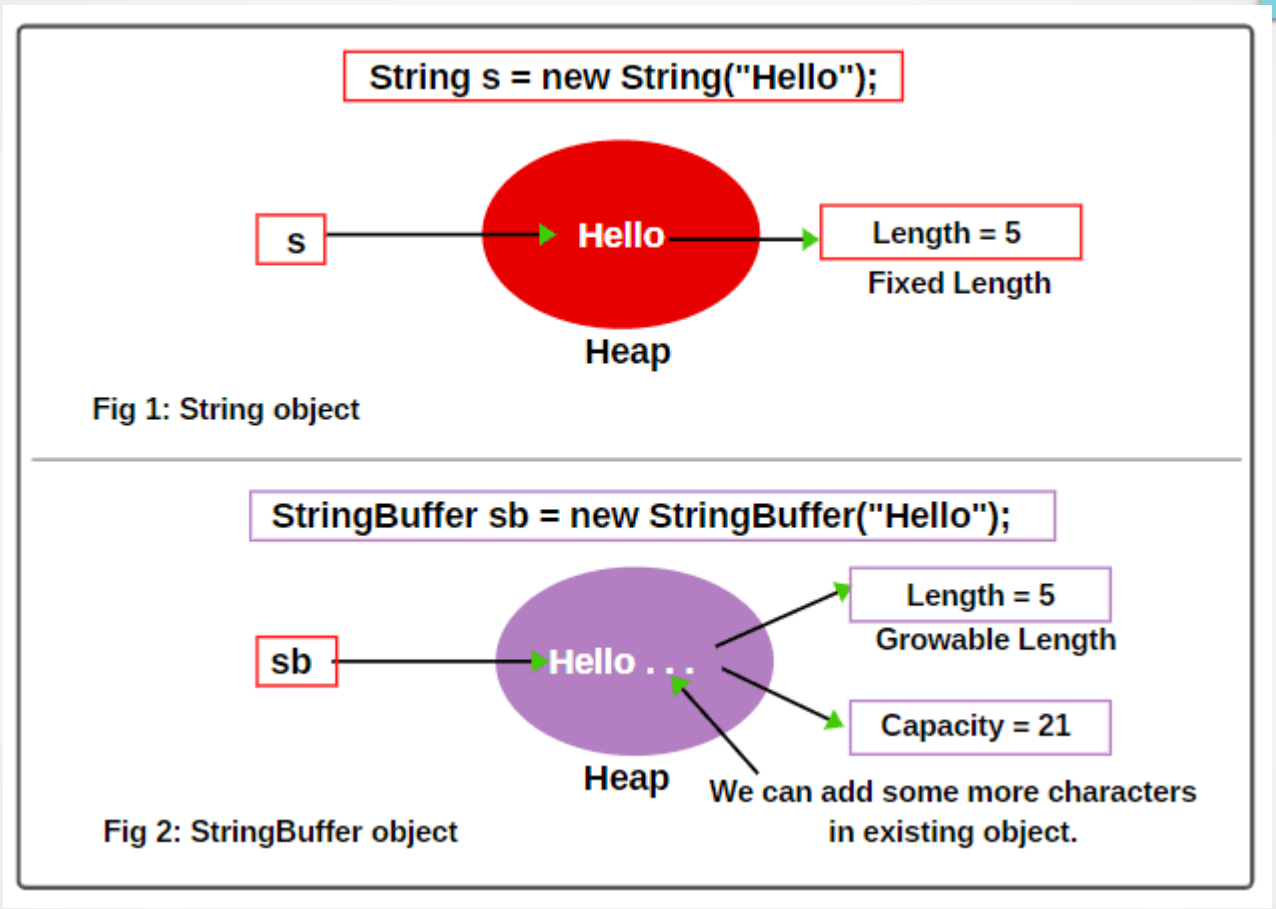

Java Heap
String s1 = “Cat”;
String s2 = “Cat”;
String s3 = new String(“Cat”);
s1 == s2; //true
s1 == s3; //false
“Cat”
“Dog”
“Cat”
String Pool


Java Heap
String s1 = " Rick " ;
String s2 = " Rick " ;
String s3 = new String(" Morty ") ;
String s3 = new String(" Rick ") ;
" Rick "
" Morty "
String Pool
" Morty "
" Rick "

StringBuffer.capacity() method

- The java.lang.StringBuffer.capacity() method returns the current capacity.
- The capacity is the amount of storage available for newly inserted characters, beyond which an allocation will occur.
- If the number of the character increases from its current capacity, it increases the capacity by (oldcapacity*2)+2.
- Syntax:

int capacity()


String s = new String("Hello");
s
Hello
Length = 5
Fixed Length
Heap
Fig 1: String object
StringBuffer sb = new StringBuffer("Hello");
sb
Hello . . .
Length = 5
Growable Length
Capacity = 21
Heap
We can add some more characters
in existing object.
Fig 2: StringBuffer object

StringBuffer.append()

- The java.lang.StringBuffer.append(String str) method appends the specified string to this character sequence.
- The characters of the String argument are appended, in order, increasing the length of this sequence by the length of the argument.
- If str is null, then the four characters "null" are appended.
- Syntax

  StringBuffer append(String str)

StringBuffer.replace()

- The java.lang.StringBuffer.replace() method replaces the characters in a substring of this sequence with characters in the specified String.
- The substring begins at the specified start and extends to the character at index end - 1 or to the end of the sequence if no such character exists.
- First the characters in the substring are removed and then the specified String is inserted at start.
- Syntax:

  StringBuffer replace(int start, int end, String str)

StringBuffer.reverse()

- The java.lang.StringBuffer.reverse() method causes this character sequence to be replaced by the reverse of the sequence.
- Syntax:

  StringBuffer reverse()

StringBuffer.charAt()

- The java.lang.StringBuffer.charAt() method returns the char value in this sequence at the specified index.
- The first char value is at index 0, the next at index 1, and so on, as in array indexing.
- The index argument must be greater than or equal to 0, and less than the length of this sequence.
- Syntax:

  char charAt(int index)

Access7.java

StringBuffer.setCharAt()

- The java.lang.StringBuffer.setCharAt() method sets the character at the specified index to ch.
- This sequence is altered to represent a new character sequence that is identical to the old character sequence, except that it contains the character ch at position index.
- Syntax:

  void setCharAt(int index, char ch)