Introduction to Java for C++ Programeers

Segment – Packages & String

By: Mahboob Ali

Packages

- A *package* is a grouping of related types providing access protection and name space management

- Create a package with a **package** statement at the top of every source file
- Use **import** statement at the beginning of the file to work with package elements
- Conventions:
  - Package names are written in all lowercase to avoid conflict with the names of classes or interfaces.
  - The beginning of the package name must be a reversed Internet domain name

    Example: **ca.senecacollege.ict**

Accessing Classes

- Same package ~ *Direct Access*
- Different package
  - Import
  - Fully-qualified class name ~ *rare!*

import Statement

```
import java.util.Scanner;


class FooClass{
    void Foo(){
        Scanner input = new Scanner(System.in);
        ……
    }
}
```

Importing Single vs Multiple classes

- Import single class
  - Explicit import (as seen in the previous example)
- Import multiple classes
  - Separate explicit imports
  - * import (i.e. java.util.*)

Explicit import or * import?

- * import can *break* code

```
import java.util.*;

import java.sql.*;

…

Date date; // form util
```

+ java.sql.Date →

```
import java.util.*;
import java.sql.*;
…
Date date; // Compiler error
```

- Better clarity with Explicit import

Fully-qualified Class Name

- Alternate to import

```
java.util.Scanner input = new java.util.Scanner(System.in);
```

Any side affects in using Import?

- None
  - Does not make your class bigger.
  - Does not affect runtime performance.
  - Saves from typing fully-qualified name ~ compiler will take care of it.
- **Java.lang** is imported by default.

Strings

- Object of class ***java.lang.String***

```
String s = new String() // empty String
String s = new String("hello");


char[] cArray = {'h','e','l','l','o'};
String s = new String(cArray);


String s = "hello"; //string literal
```

Not recommended

recommended

- String classes uses ***character array*** to store text.
- String is sequence of Unicode characters.
- String is ***immutable***.
- String pool ~ saves memory

Common operation

- Comparing.
- Searching.
- Examining individual characters.
- Extracting sub strings.
- Case translation.
- Replace.
- Split.

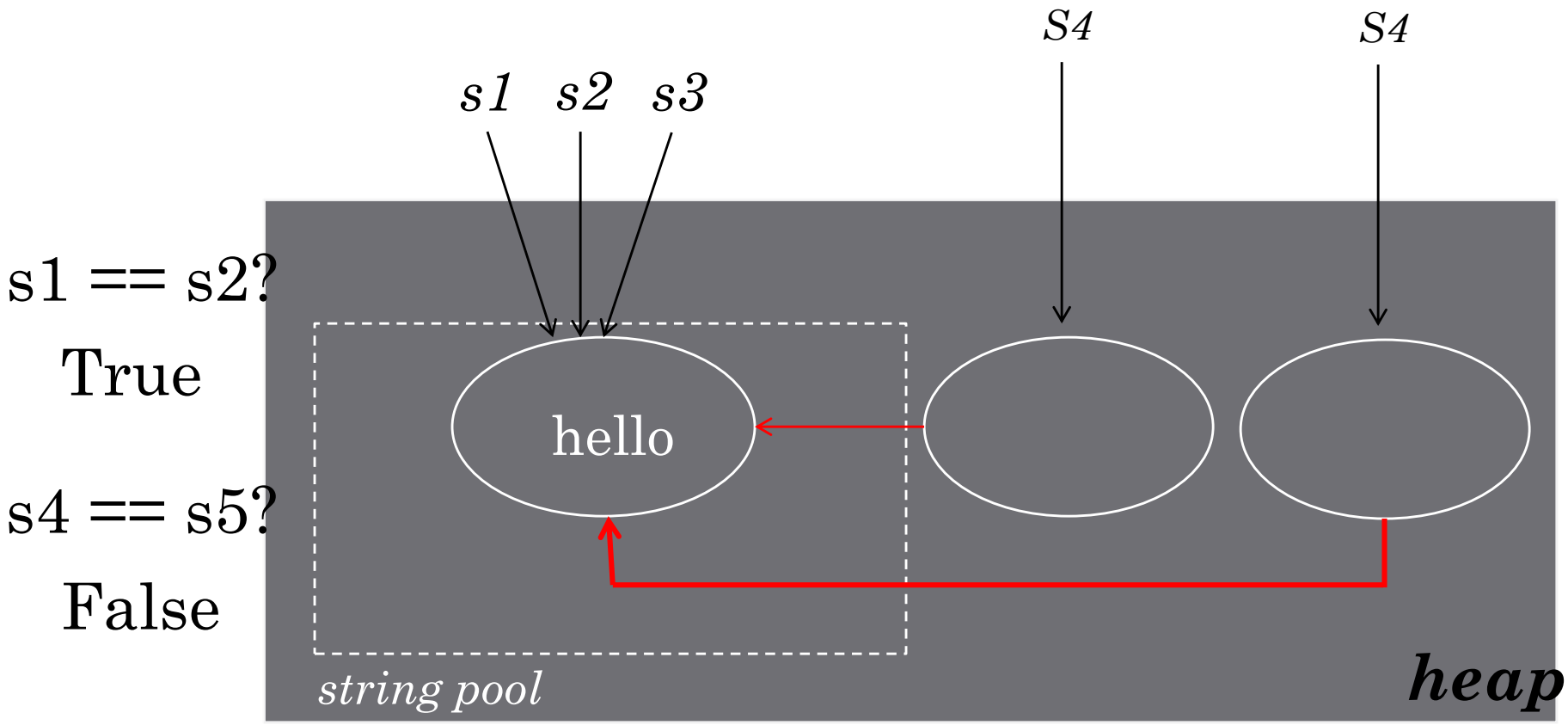
String Pool
String literal vs Using **new**

- String (via *string literal*)
  - Stored in *string pool* on heap.
  - Literals with same content *share same storage*.
- String (via ***new***)
  - Same as regular object.
  - No storage sharing.

String *s1* = “hello”;
String *s2* = “hello”;
String *s3* = *s1;*

String *s4* = **new** String(“hello”);
String *s5* = **new** String(“hello”);

Advantage = Saves memory



s1 == s2?

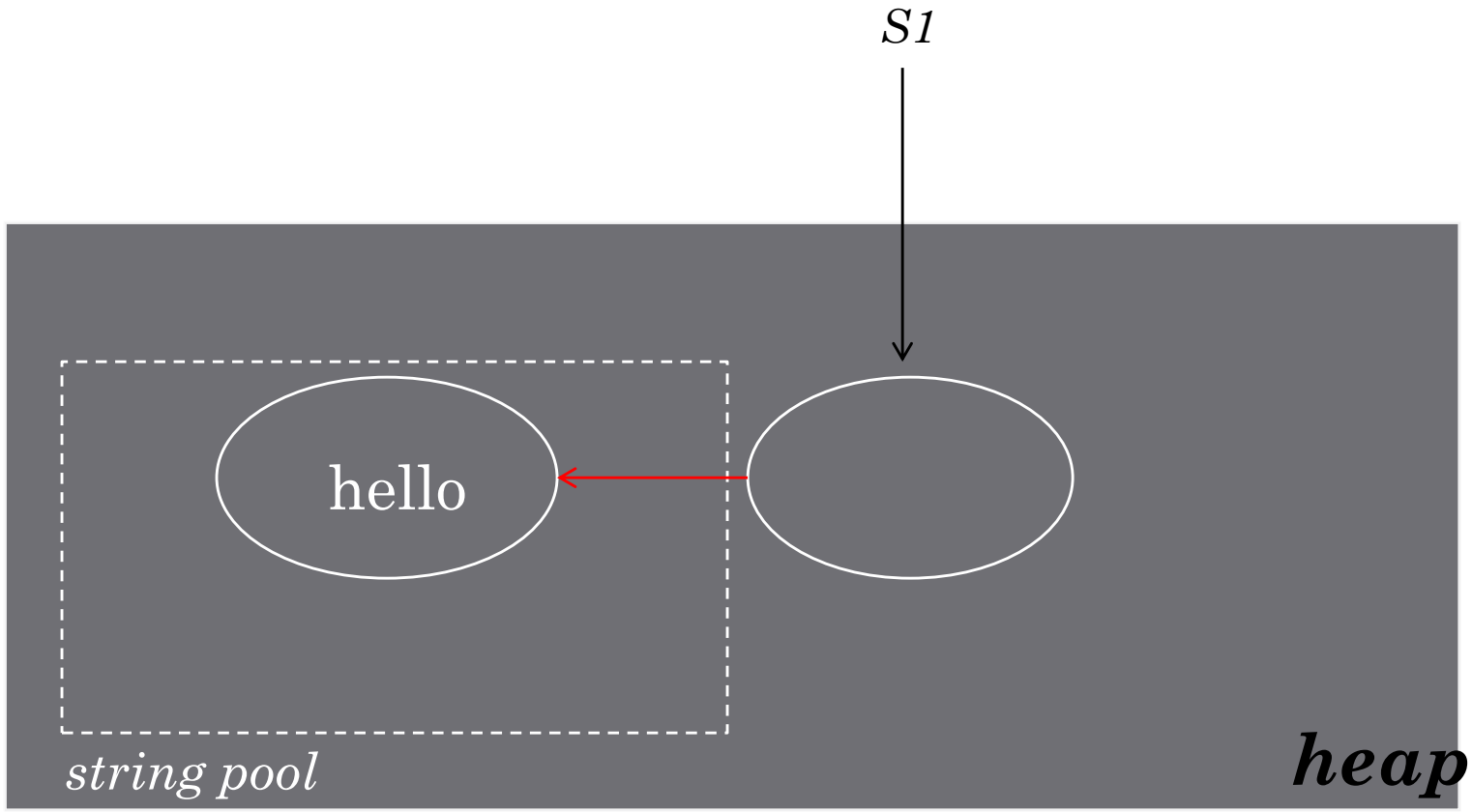
True

s4 == s5?

False

String *s1* = **new** String(“hello”);

How it will be created?



Class StringBuilder

- Creating and manipulating strings in ***dynamic*** way.
  - In other words modifiable strings.
- How?
  - Every `StringBuilder` is capable of storing a number of characters specified by its capacity.
  - If the capacity increases it expands itself.
- Syntax:

```
StringBuilder sb = new StringBuilder();

sb.append("Greetings");
```

  - The following syntax will produce a string builder with the length of 9 and capacity of 16.