innovate achieve lead

BITS Pilani
Pilani Campus

Reference Types & Java Memory Management

Stack and Heap

- When your program is running, some memory is used to store local variables. This memory is known as the stack.
- We can use a table to represent variables stored on the stack

Var	Value
x	42
y	3.7

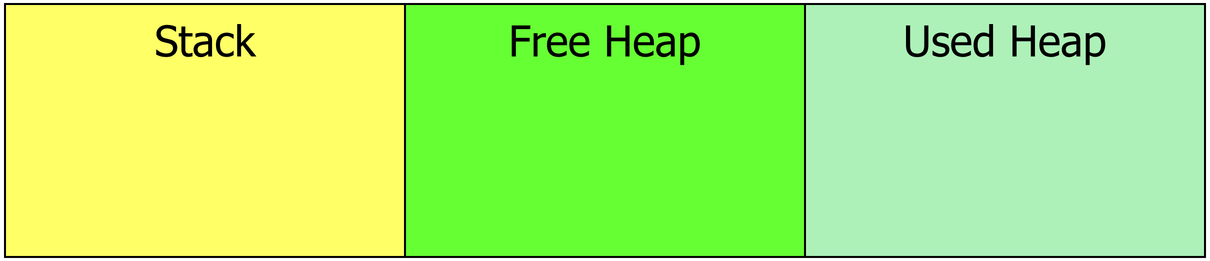
- The rest of memory is known as the heap and is used for dynamically allocated “stuff” (recall using malloc() in C)

Main Memory

The stack grows and shrinks as needed (why?)

The used part of the heap ("allocated") also grows and shrinks.

Some of the heap is unused ("free")



Stack

Free Heap

Used Heap

Object Creation

Consider this code that creates two strings

```
String s1, s2;
s1 = new String( “abc” );
s2 =  “abc” ;
```

Note: Because Strings are very common, using new when creating String objects is optional.

Where are these variables and object located in memory?

Why do we care?

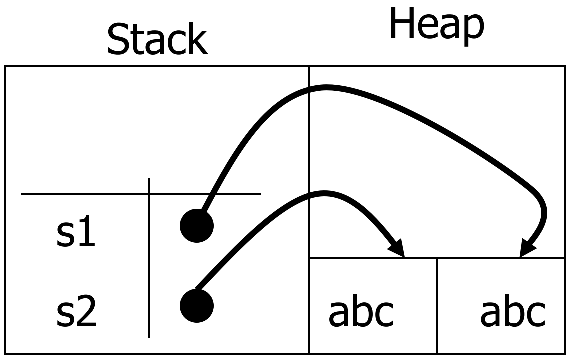
Objects in Memory

The statement `String s1, s2;` creates two local variables on the stack. The statements

```
        s1 = new String( "abc" );
        s2 = "abc";
```

create objects on the heap. `s1` and `s2` contain the memory addresses of these objects giving us the picture of memory shown below.

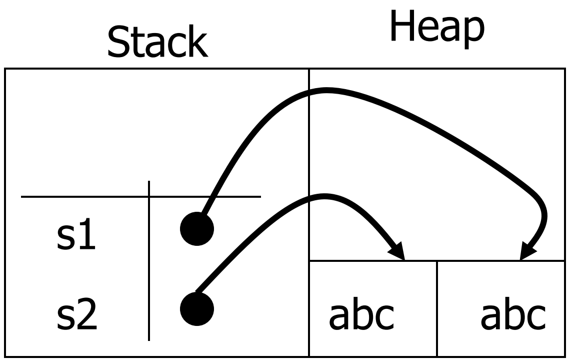
`s1` and `s2` are called reference variables. Reference variables which do not contain the memory address of any object contain the special value null



Why We Care (1 of 4)

Given the previous code

```
   String s1, s2;
   s1 = new String( "abc" );
   s2 = "abc";
```



and corresponding picture of memory consider the expression `s1 == s2`

Recall that `s1` and `s2` contain the addresses of their respective String objects. Since the String objects have different addresses on the heap, `s1 == s2` is ***false***. The == operator determines if two reference variables refer to the same Object.

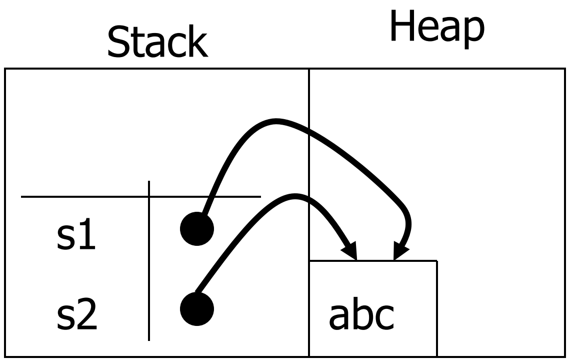
So how do we compare Strings for equality?
Strings (and other objects) implement a method named `equals`.
To check if two Strings are the same, use the expression

```
s1.equals( s2 );
```

Why We Care (2 of 4)

On the other hand, consider this code and corresponding picture of memory

```
String s1 = "abc";
String s2 = s1;
```



s1 and s2 are ALIASed

Now s1 and s2 refer to the same String object. This is known as ALIASING, is often unintentional, and can be dangerous.

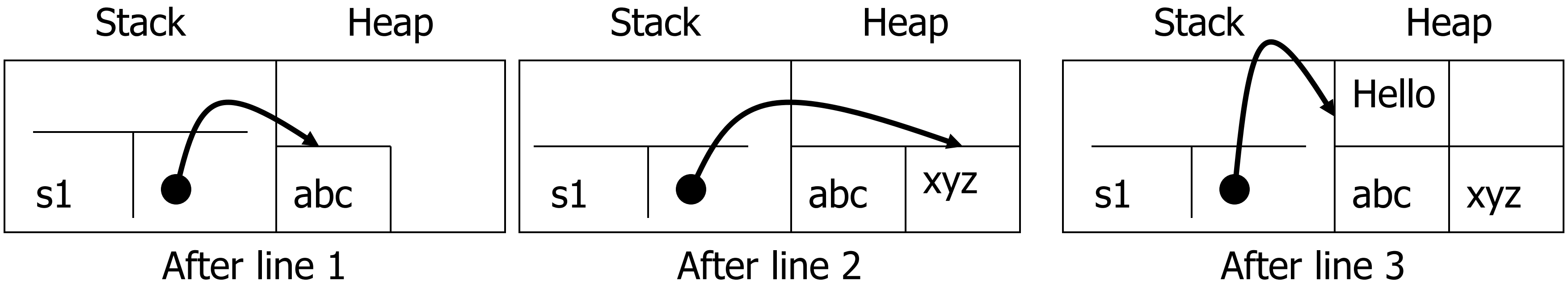
If your intent is for s2 to be a copy of s1, then the correct code is

```
String s2 = new String( s1 );
```

Why We Care (3 of 4)

Consider this code and the changing picture of memory

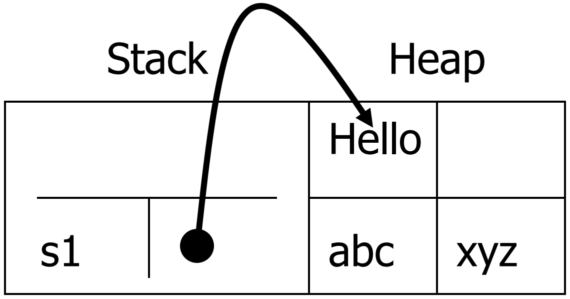
```
String s1 = “abc”;    // line 1
s1 = “xyz”;           // line 2
S1 = “Hello”;         // line 3
```



Why We Care (4 of 4)

- Garbage collection

As the diagram shows, after line 3 is executed no variable refers to the String objects which contain “abc” or “xyz”.

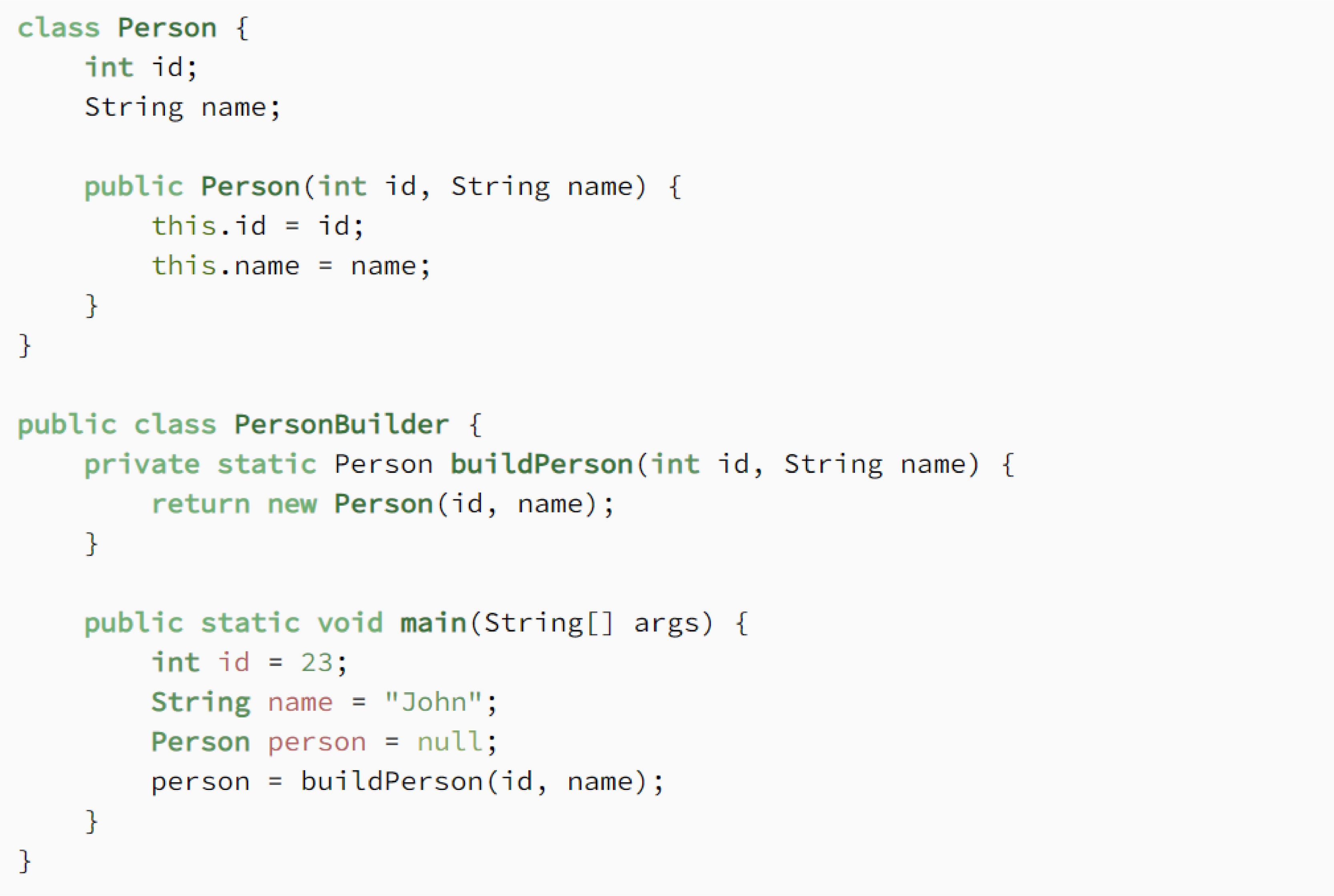


After line 3

In C/C++, we’d consider this a “memory leak”. In C/C++ it’s the programmer’s responsibility to return dynamically allocated memory back to the free heap. (recall using **malloc** and **free** in C?)
Not so in Java!

Java has a built-in “garbage collector”. From time to time Java detects objects has been “orphaned” because no reference variable refers to them. The garbage collector automatically returns the memory for those objects to the free heap.

THANK YOU

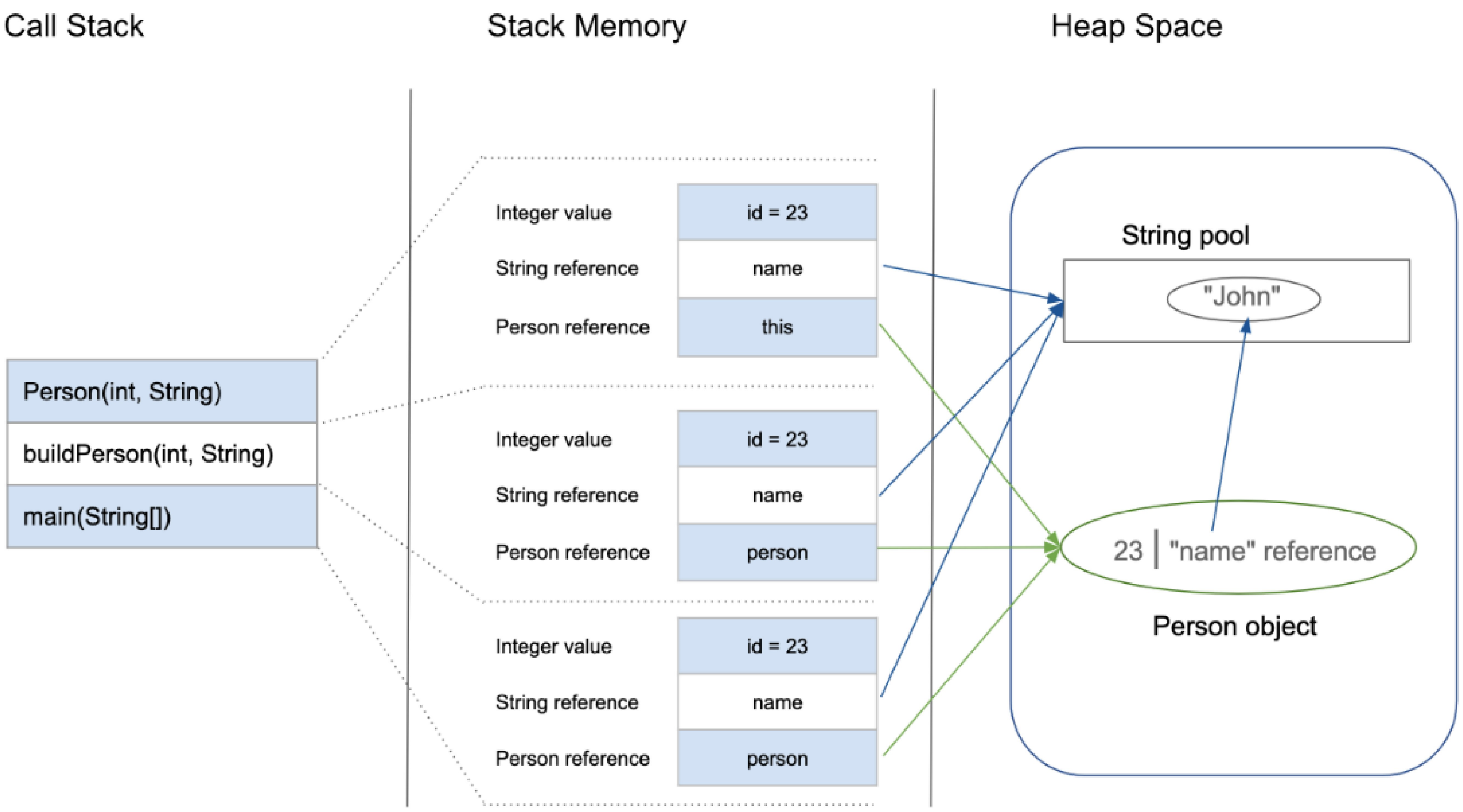


```java
class Person {
    int id;
    String name;

    public Person(int id, String name) {
        this.id = id;
        this.name = name;
    }
}

public class PersonBuilder {
    private static Person buildPerson(int id, String name) {
        return new Person(id, name);
    }

    public static void main(String[] args) {
        int id = 23;
        String name = "John";
        Person person = null;
        person = buildPerson(id, name);
    }
}
```


Call Stack
Stack Memory
Heap Space
Person(int, String)
buildPerson(int, String)
main(String[])
Integer value
id = 23
String reference
name
Person reference
this
Integer value
id = 23
String reference
name
Person reference
person
Integer value
id = 23
String reference
name
Person reference
person
String pool
"John"
23 | "name" reference
Person object