
ORACLE®
ACADEMY

Java Foundations

7-2
Instantiating Objects

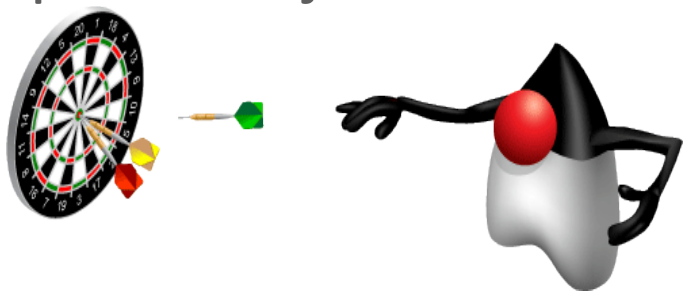
ORACLE ACADEMY

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Objectives

This lesson covers the following objectives:

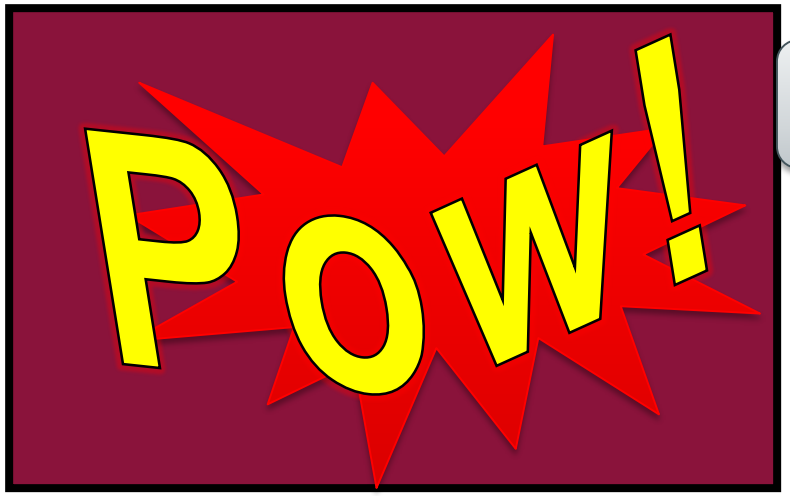
- Understand the memory consequences of instantiating objects
- Understand object references
- Understand the difference between stack and heap memory
- Understand how `Strings` are special objects

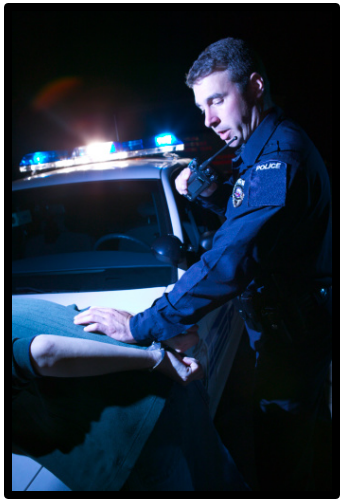


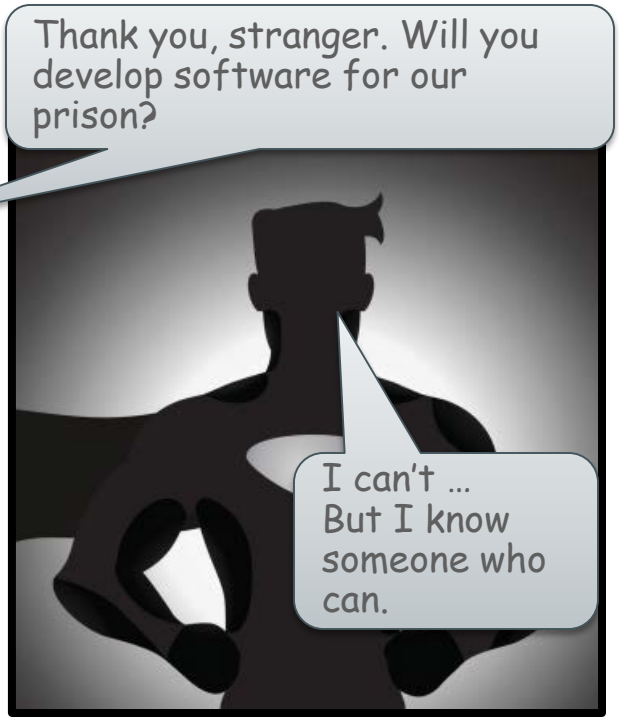
Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.



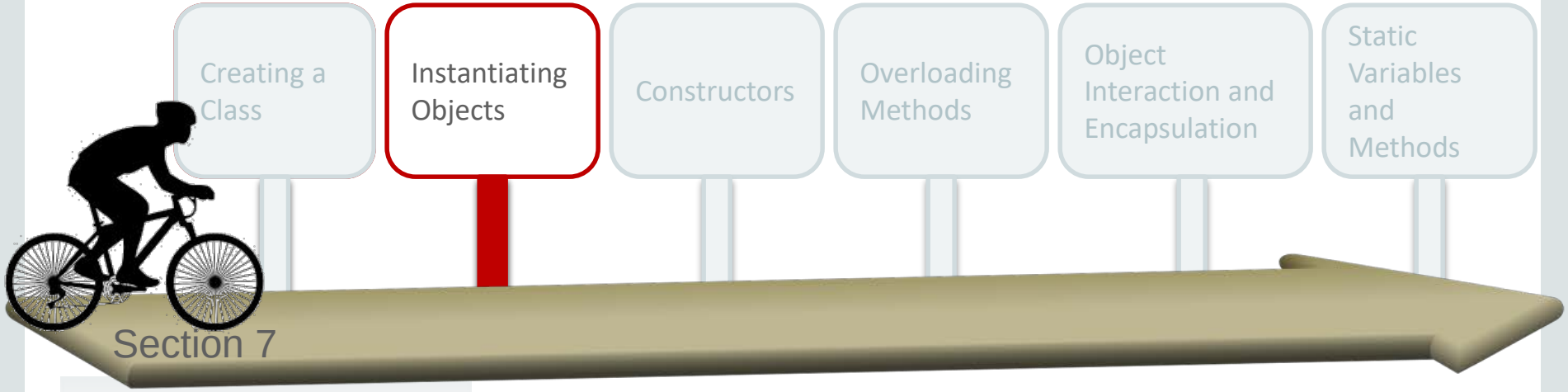




Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Topics

- Objects in Memory
- Object References and Memory Management
- Instantiating `Strings`



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

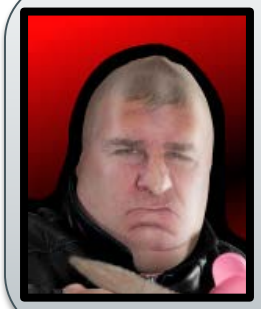
Describing a Prisoner

- Properties:
  - Name
  - Height
  - Years Sentenced
- Behaviors:
  - Think about what they've done

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Exercise 1, Part 1

- Create a new Java project.
- Create a `PrisonTest` class with a main method.
- Create a `Prisoner` class based on the description in the previous slide.
- Instantiate two prisoners and assign them the following properties:



```
Variable:  bubba
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years
```

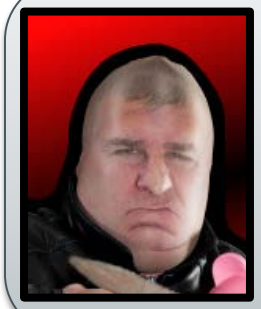
```
Variable:  twitch
Name:      Twitch
Height:    5'8"   (1.73m)
Sentence:  3 years
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Exercise 1, Part 2

Can prisoners fool security by impersonating each other?

- Write a print statement with a boolean expression that tests if `bubba == twitch`.
- Change the properties of `twitch` so that they match `bubba`.
- Then test the equality of these objects again.



```
Variable:  bubba
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years
```

```
Variable:  twitch
Name:      Bubba
Height:    6'10"   (2.08m)
Sentence:  4 years
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Programming the Prisoner Class

Your class may look something like this:

```
public class Prisoner {
    public String name;
    public double height;
    public int sentence;

    public void think(){
        System.out.println("I'll have my revenge.");
    }
}
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Prisoner Impersonation

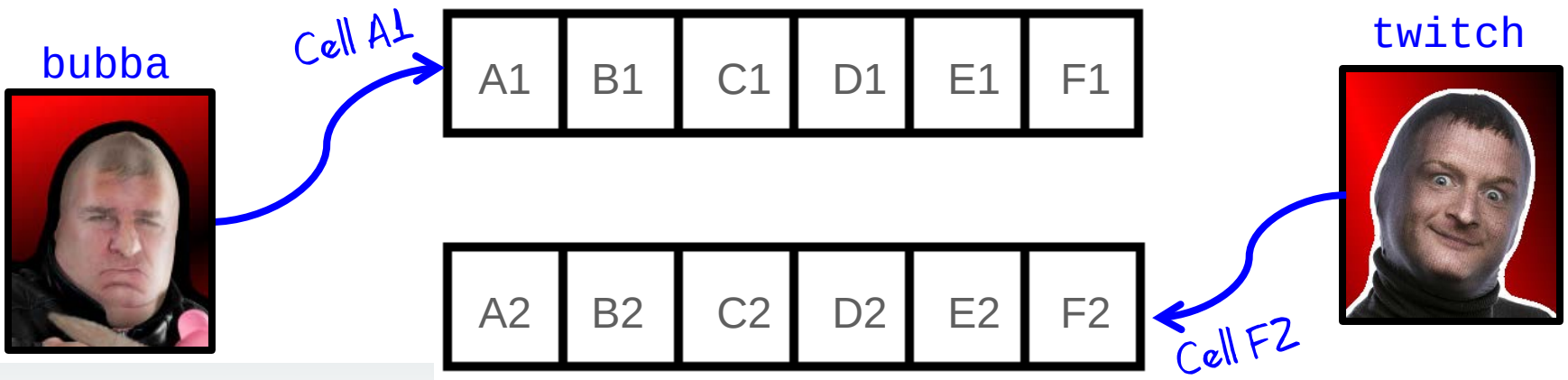
- The boolean `bubba == twitch` is `false.`
  - Security wasn't fooled by prisoners who share the same properties.
  - Security understood that each prisoner was a unique object.
- How is this possible?

```
public class PrisonTest {
    public static void main(String[] args){
        Prisoner bubba = new Prisoner();
        Prisoner twitch = new Prisoner();
        …
        System.out.println(bubba == twitch);      //false
    }
}
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Prisoner Locations

- Prisoners live in cells.
- New prisoners are assigned an available cell for living quarters.
- If a prisoner lives in a unique cell, he's a unique object.

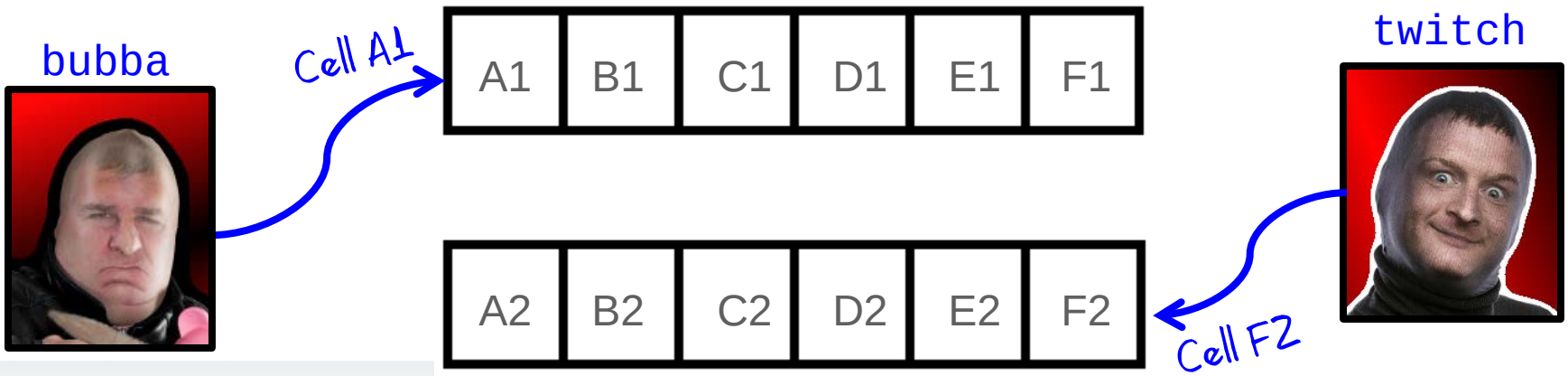


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Prisoner Object Locations

- Cells are like locations in memory.
- Instantiating a `Prisoner` fills an available location in memory with the new `Prisoner` object.

```
public class PrisonTest {
    public static void main(String[] args){
        Prisoner bubba = new Prisoner();
        Prisoner twitch = new Prisoner();
    }
}
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

The new Keyword

- The new keyword allocates available memory to store a newly created object.
- Java developers don't need to know an object's location in memory.
  - We only need to know the variable for the object.
  - But we can still print memory addresses.

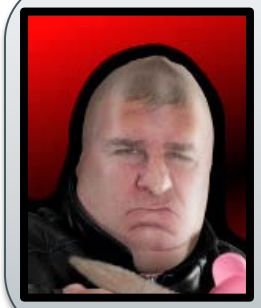
```
public class PrisonTest {
    public static void main(String[] args){
        Prisoner bubba = new Prisoner();
        Prisoner twitch = new Prisoner();
        System.out.println(bubba);      //prisontest.Prisoner@15db9742
        System.out.println(twitch);     //prisontest.Prisoner@6d06d69c
    }
}
```

Memory addresses

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Objects with the Same Properties

- Objects may share the same properties.
- But it doesn't mean that these objects are equal.
- As long as you use the new keyword during instantiation …
  - You'll have unique objects.
  - Each object will have a different location in memory.



```
Variable:  bubba
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years

Memory Address: @15db9742
```

```
Variable:  twitch
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years

Memory Address: @6d06d69c
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Comparing Objects

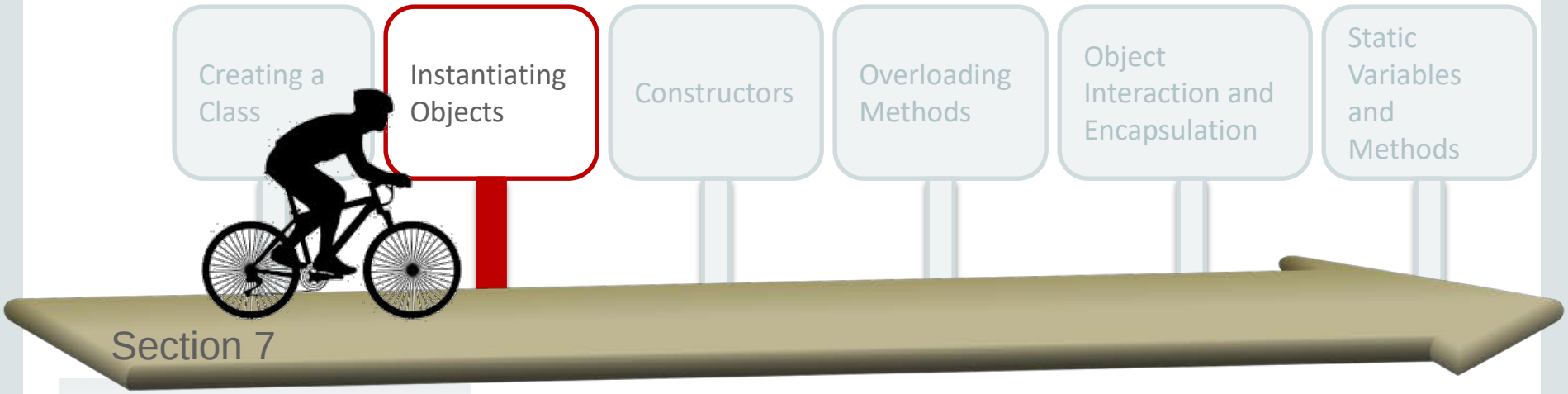
- If you compare two objects using the == operator …
  - You're checking if their **memory addresses** are equal.
  - You're **not** checking if their fields are equal.
- The boolean `bubba == twitch` is `false` because …
  - Memory addresses `@15db9742` and `@6d06d69c` are different.
  - It doesn't matter if `bubba` and `twitch` share the same properties.

```
public class PrisonTest {
    public static void main(String[] args){
        Prisoner bubba = new Prisoner();
        Prisoner twitch = new Prisoner();
        …
        System.out.println(bubba == twitch);      //false
    }
}
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

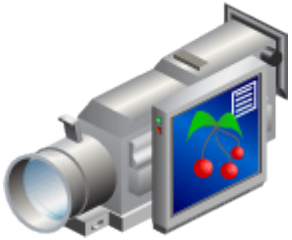
Topics

- Objects in Memory
- Object References and Memory Management
- Instantiating `Strings`



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Accessing Objects by Using a Reference



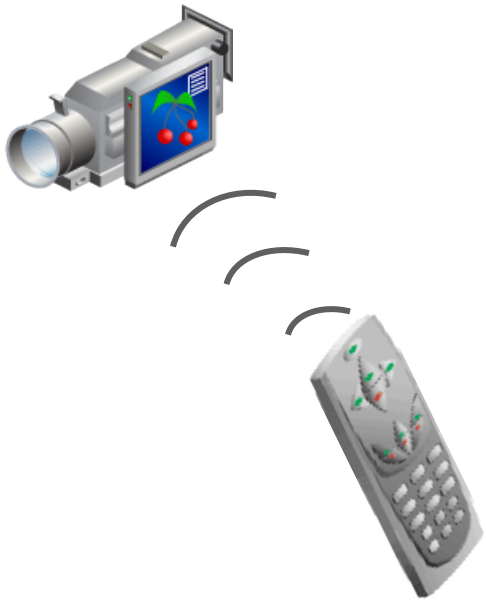
The camera is like the **object** that's accessed by using a reference.

The remote is like the **reference** that's used to access the camera.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Working with Object References

1 Pick up remote to gain access to the camera.



2 Press remote controls to have the camera do something.

1 Create a Camera object and get a reference to it.

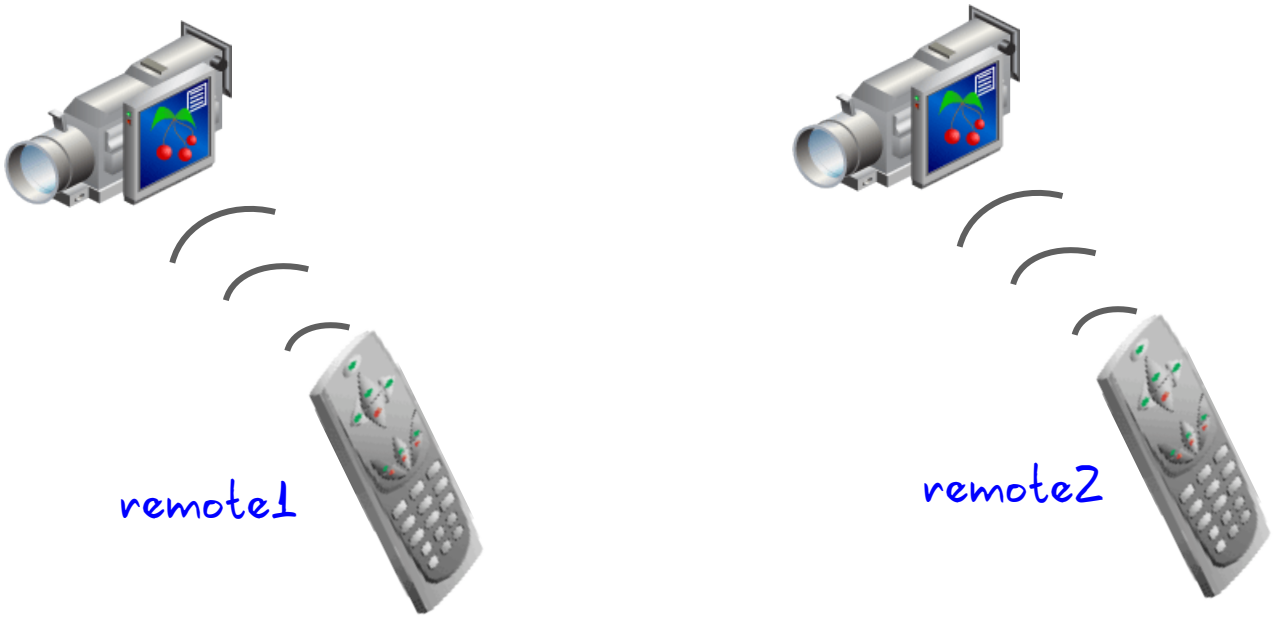
```
Camera remote1 = new Camera();
```

2 Call a method to have the Camera object do something.

```
remote1.play();
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Working with Object References: Example 1



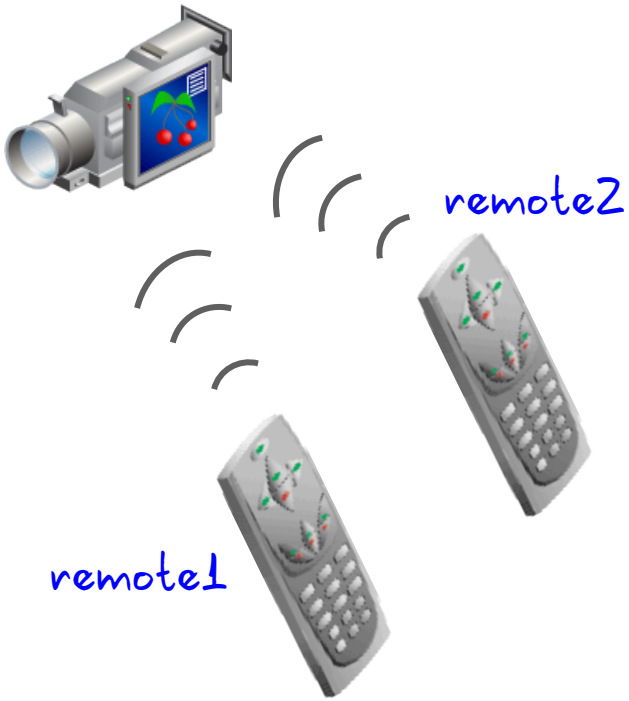

```
Camera remote1 = new Camera();
Camera remote2 = new Camera();

remote1.play();
remote2.play();
```

There are two Camera objects.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Working with Object References: Example 2



There's only one Camera object.

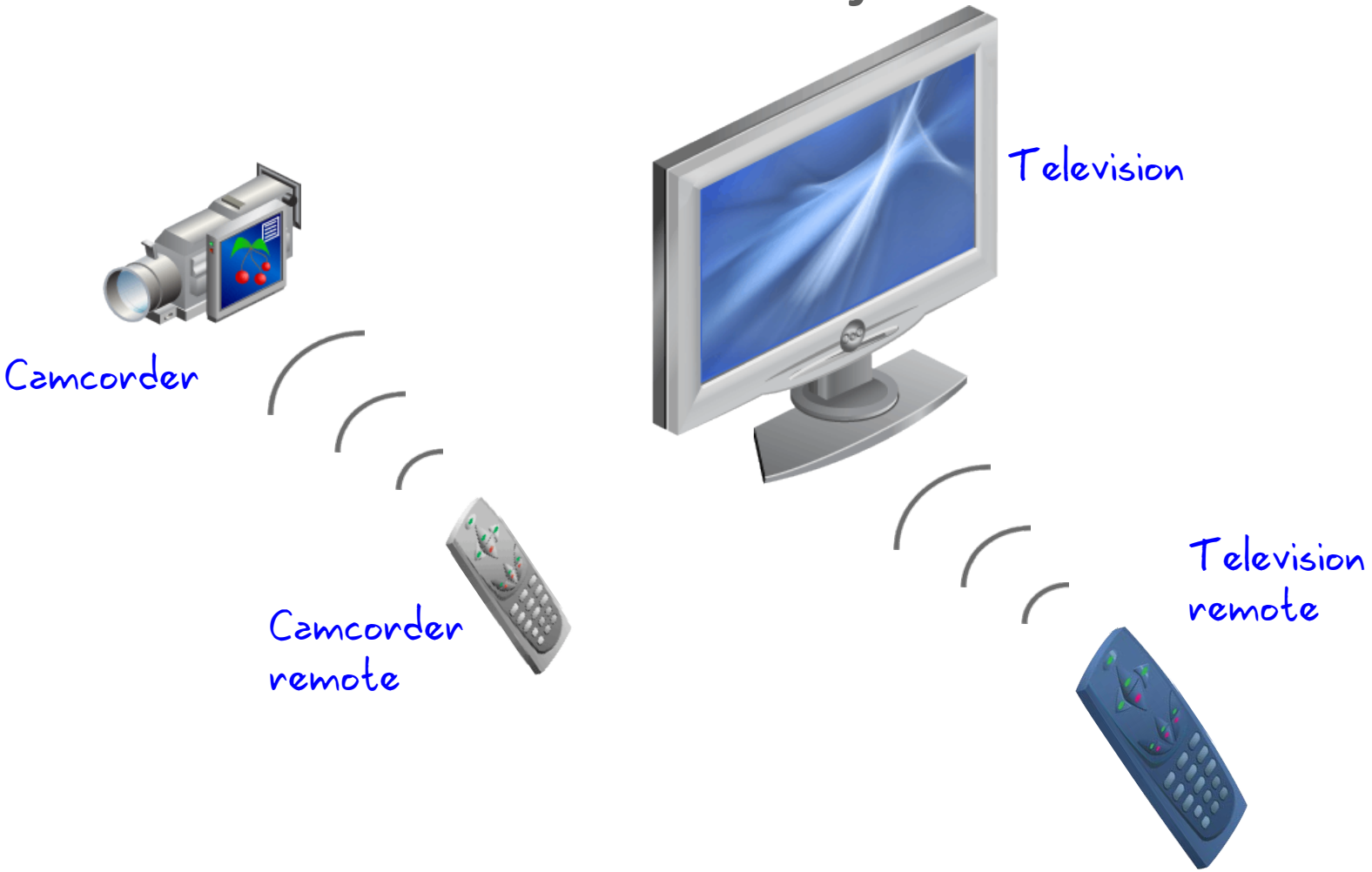
```
Camera remote1 = new Camera();

Camera remote2 = remote1;

remote1.play();
remote2.stop();
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

References to Different Objects



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

References to Different Objects: Example


```
Camera remote1 = new Camera();
remote1.menu();

TV remote2 = new TV();
remote2.menu();

Prisoner bubba = new Prisoner();
bubba.think();
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

References to Different Objects: Example

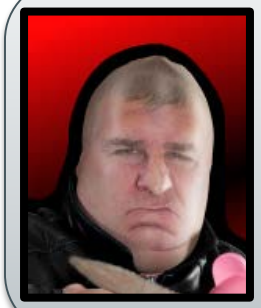
- The following example isn't allowed because …
  - The **Reference Type** doesn't match the **Object Type.**
  - A prisoner and a TV are completely different things.

```
Prisoner twitch = new TV();
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Exercise 2

- Continue experimenting with the `PrisonTest` class.
- Is security fooled when reference variables change?
  - Instantiate two prisoners and assign them the properties below.
  - Test the equality of these objects.
  - Then set the reference variable for `bubba` equal to `twitch`.
  - Test the equality of these objects again.



```
Variable:  bubba
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years
```

```
Variable:  twitch
Name:      Twitch
Height:    5'8"   (1.73m)
Sentence:  3 years
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Stack Memory and Heap Memory

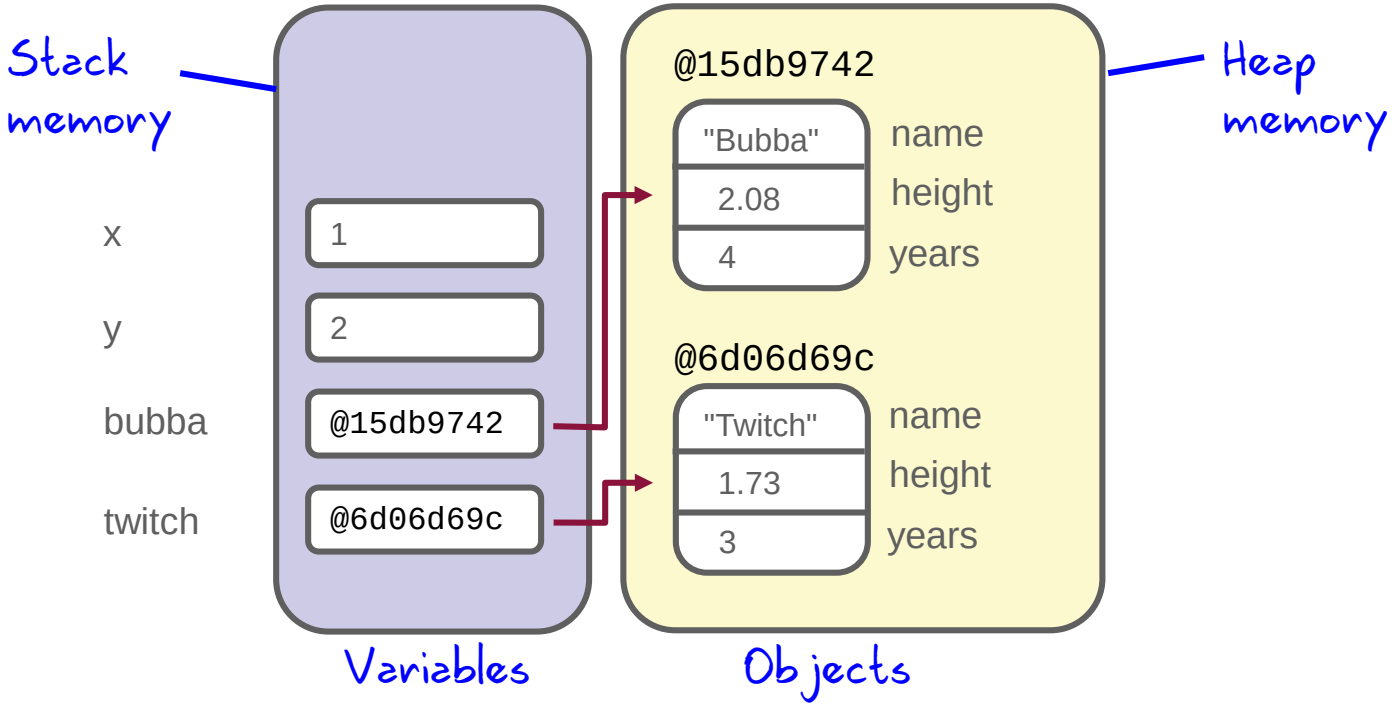
Understanding the results of Exercise 2 requires an understanding of the types of memory that Java uses.

- **Stack memory** is used to store …
  - Local variables
  - Primitives
  - References to locations in the heap memory
- **Heap memory** is used to store …
  - Objects

Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

References and Objects in Memory

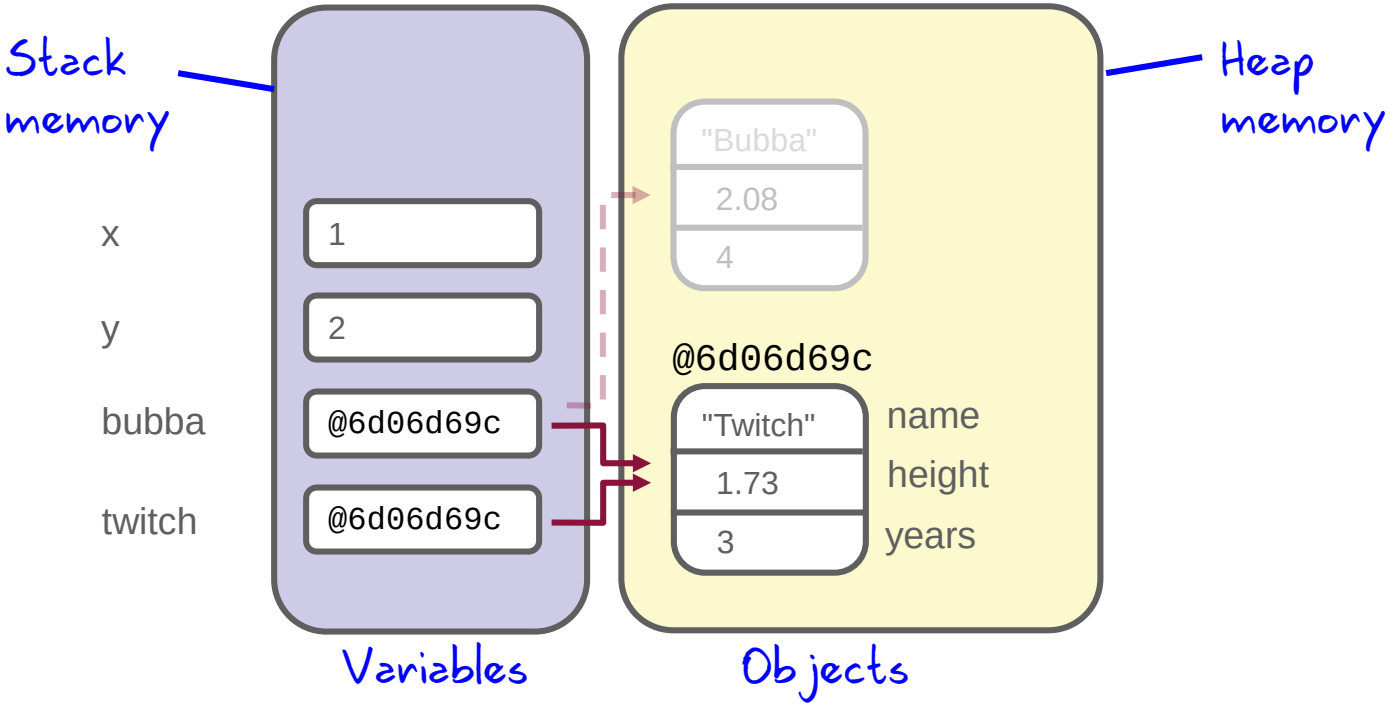
```
int x = 1;
int y = 2;
Prisoner bubba = new Prisoner();
Prisoner twitch = new Prisoner();
…
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Assigning a Reference to Another Reference

```
bubba = twitch;
```



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Two References, One Object

- As of line 14, `bubba` and `twitch` reference the same object.
- Either reference variable could be used to access the same data.

```
11 Prisoner bubba = new Prisoner();
12 Prisoner twitch = new Prisoner();
13
14 bubba = twitch;
15
16 bubba.name = "Bubba";
17 twitch.name = "Twitch";
19
20 System.out.println(bubba.name);       //Twitch
21 System.out.println(bubba == twitch); //true
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Two References, Two Primitives

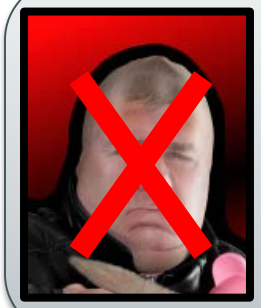
- Primitives are always separate variables.
- Primitive values always occupy different locations in the stack memory.
- Line 14 briefly makes primitive values x and y equal.

```
11 int x;
12 int y;
13
14 x = y;
15
16 x = 1;
17 y = 2;
19
20 System.out.println(x);              //1
21 System.out.println(x == y);         //false
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

What Happened to Bubba?

- If no more reference variables point to an object …
- Java **automatically** clears the memory once occupied by that object.
  - This is called **Garbage Collection.**
  - The data associated with this object is lost forever.



```
Variable:
Name:       Bubba
Height:     6'10"   (2.08m)
Sentence:   4 years

Memory Address:
```

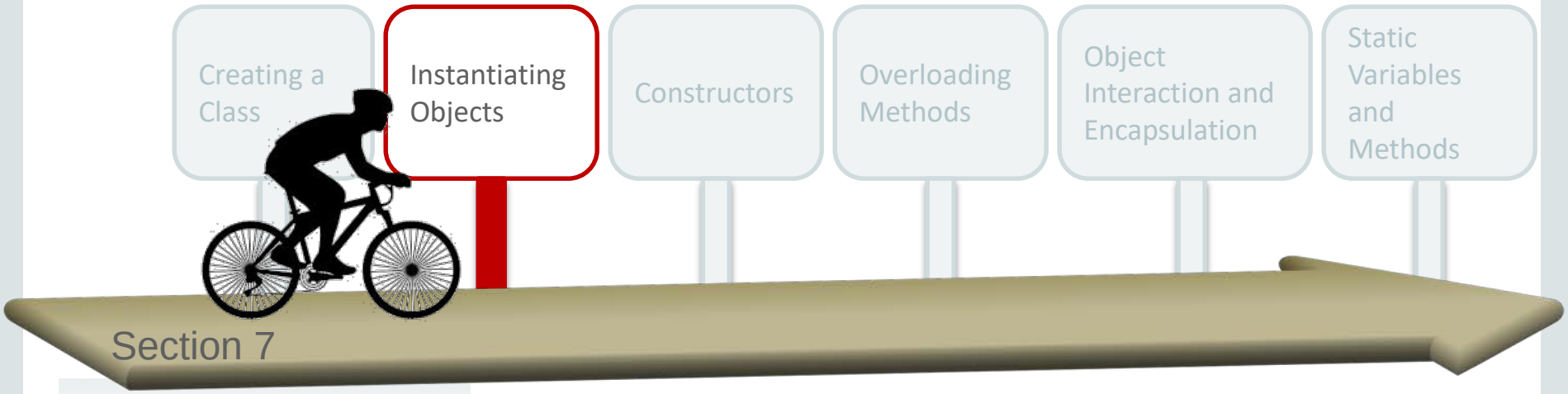
```
Variables: twitch, bubba
Name:       Twitch
Height:     5'8"    (1.73m)
Sentence:   3 years

Memory Address: @6d06d69c
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Topics

- Objects in Memory
- Object References and Memory Management
- Instantiating `Strings`



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Strings Are Special Objects

- Printing a String reference prints the actual String instead of the object's memory address.
- Strings can be instantiated with the new keyword.
  - But you shouldn't do this.

```
String s1 = new String("Test");
```

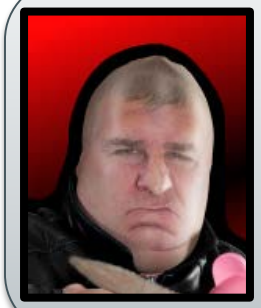
- Strings should be instantiated without new.
  - This is more memory-efficient.
  - We'll explore why in the next few slides.

```
String s2 = "Test";
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Exercise 3

- Continue experimenting with the `PrisonTest` class.
- See the memory consequences of `Strings` for yourself.
  - Instantiate two prisoners with the names shown below.
  - Set their names by using the `new` keyword and test the equality of these `Strings` by using ==.
  - Set their names without using the `new` keyword and test the equality of these `Strings` by using ==.



```
Variable:  bubba
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years
```

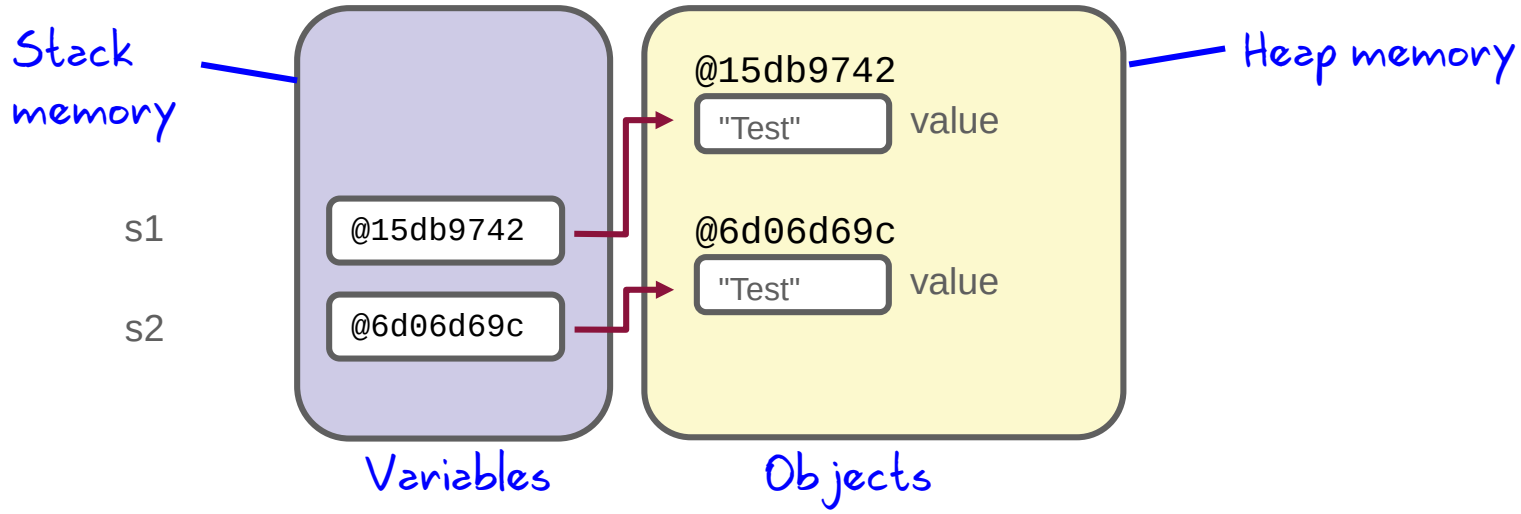
```
Variable:  twitch
Name:      Bubba
Height:    6'10"  (2.08m)
Sentence:  4 years
```


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Instantiating `Strings` with the new Keyword

Using the new keyword creates two different references to two different objects.

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

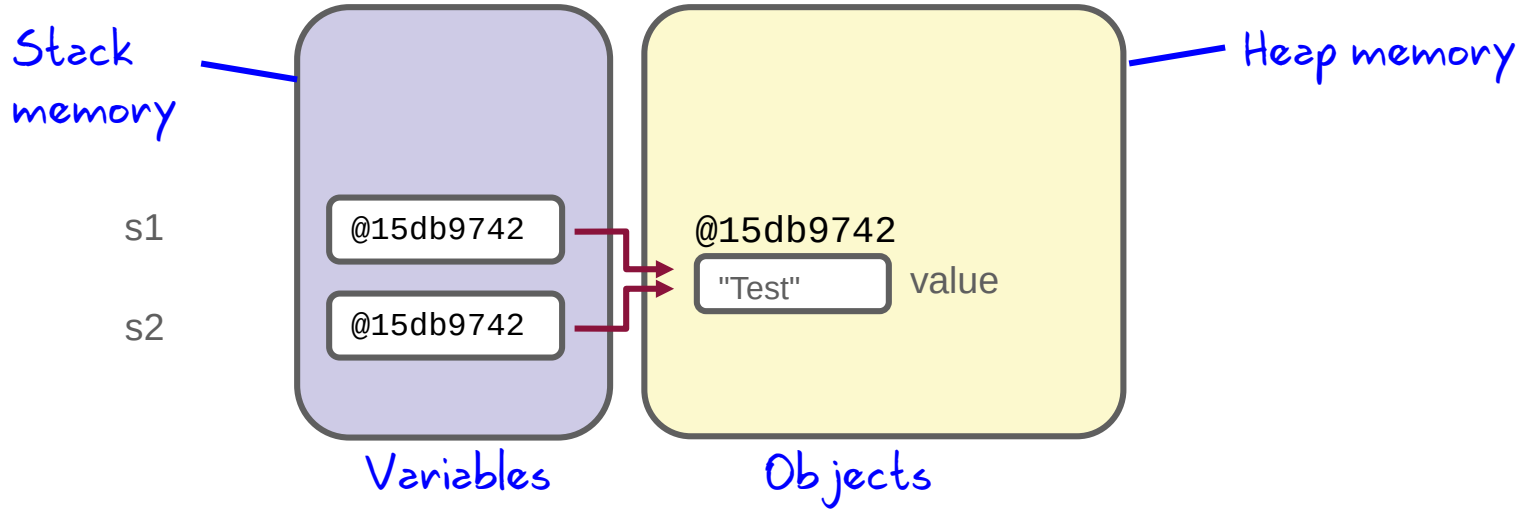


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Instantiating `Strings` Without the `new` Keyword

- Java automatically recognizes identical `Strings` and saves memory by storing the object only once.
- This creates two different references to one object.

```
String s1 = "Test";
String s2 = "Test";
```

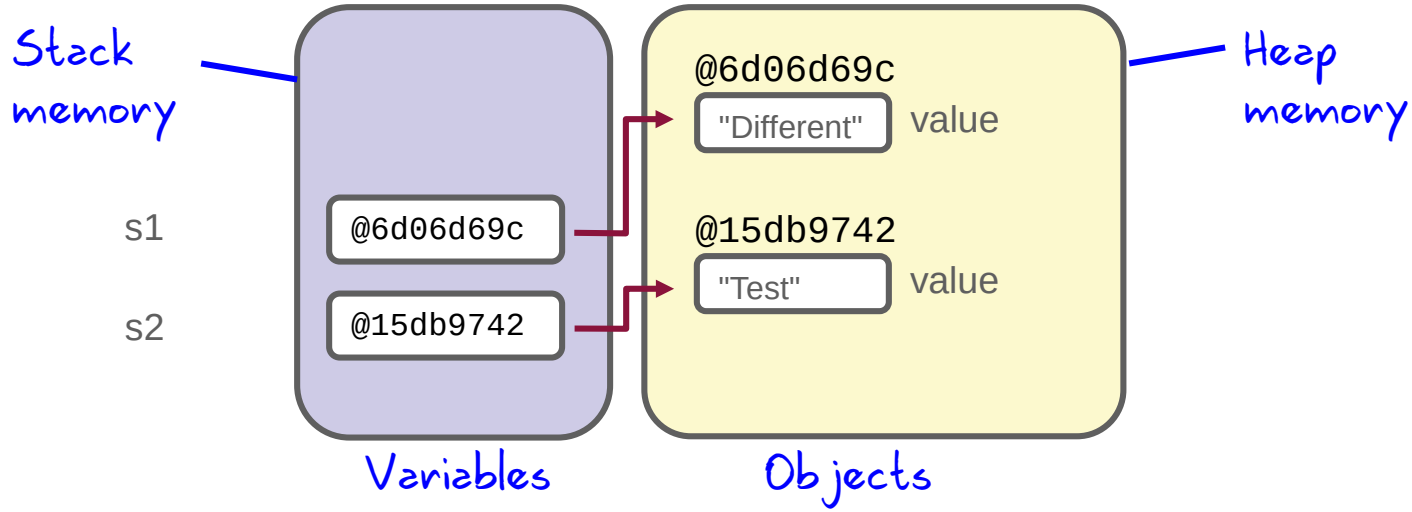


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

`String` References

- Altering a `String` using one reference won't affect other references.
- Java allocates new memory for a different `String`.

```
String s1 = "Test";
String s2 = "Test";
s1 = "Different";
```

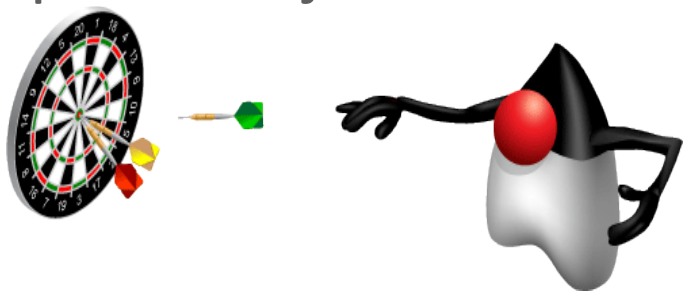


Copyright © 2017, Oracle and/or its affiliates. All rights reserved.

Summary

In this lesson, you should have learned how to:

- Understand the memory consequences of instantiating objects
- Understand object references
- Understand the difference between stack and heap memory
- Understand how `Strings` are special objects



Copyright © 2017, Oracle and/or its affiliates. All rights reserved.


ORACLE
ACADEMY