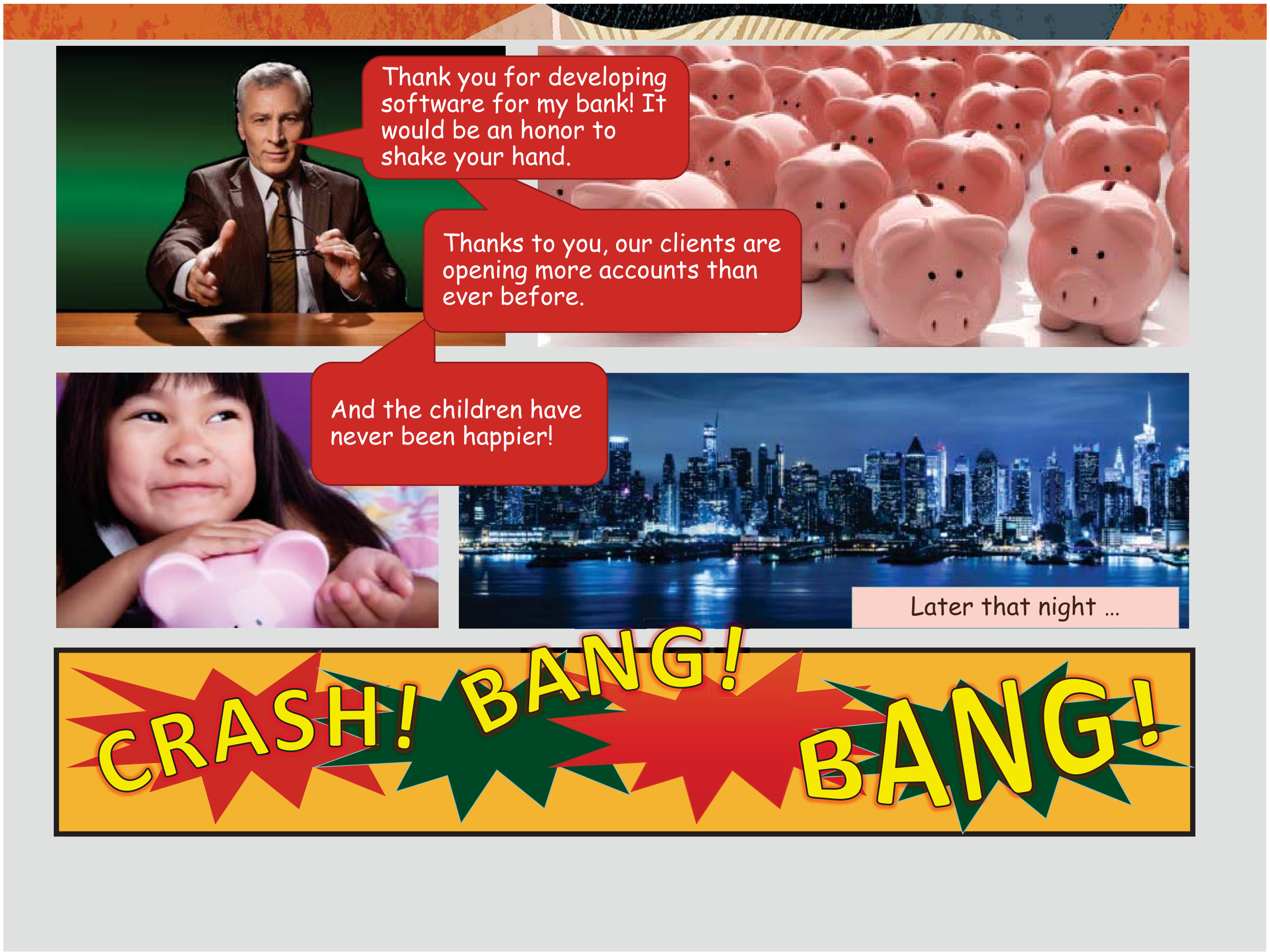
Java Foundations

Instantiating Objects

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


Thank you for developing software for my bank! It would be an honor to shake your hand.
Thanks to you, our clients are opening more accounts than ever before.
And the children have never been happier!
Later that night ...
CRASH! BANG!
BANG!

BANG!
Ha! Ha! Ha! Stealing is fun!

Pow!
Halt, thieves!
Thank you, stranger. Will you develop software for our prison?
I can't ... But I know someone who can.

Describing a Prisoner

- Properties:
  - Name
  - Height
  - Years Sentenced

- Behaviors:
  - Think about what they've done

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

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

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.");
   }//end method think
}//end class Prisoner
```

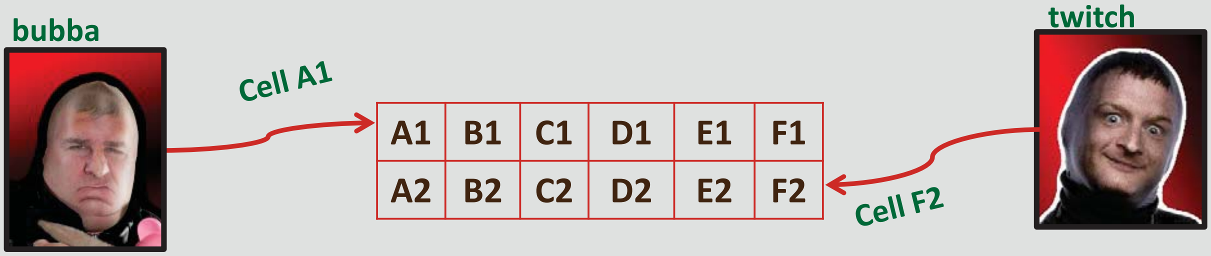
Prisoner Impersonation

- 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
   }//end method main
}//end class PrisonTest
```

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

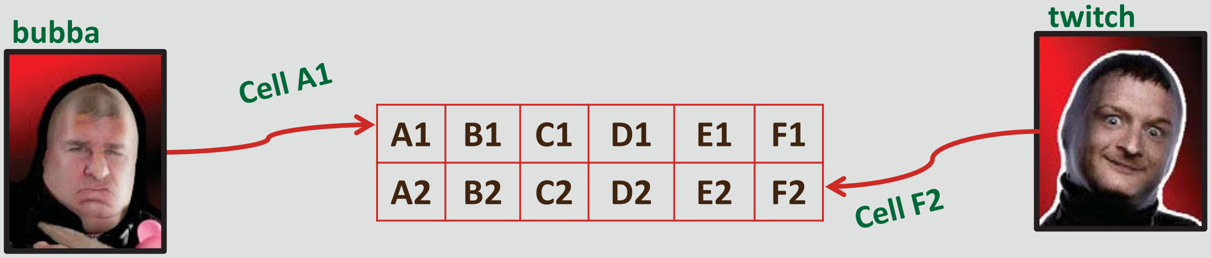


A1	B1	C1	D1	E1	F1
A2	B2	C2	D2	E2	F2

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();
   }//end method main
}//end class PrisonTest
```



A1	B1	C1	D1	E1	F1
A2	B2	C2	D2	E2	F2

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

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
    }//end method main
}//end class PrisonTest
```

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
```

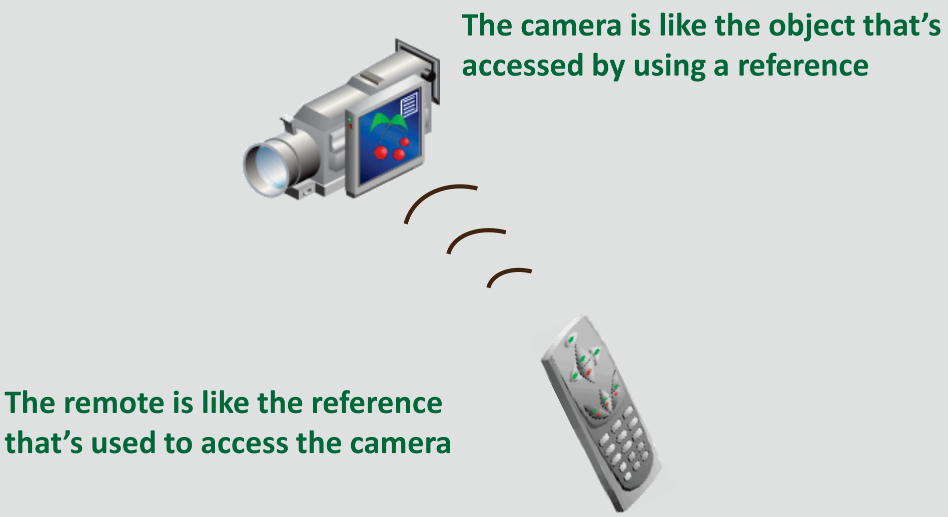
```
Variable:   twitch
Name:       Bubba
Height:     6'10"
           (2.08m)
Sentence:   4 years
Memory Address
:@6d06d69c
```

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

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

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

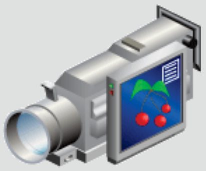
Working with Object References

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

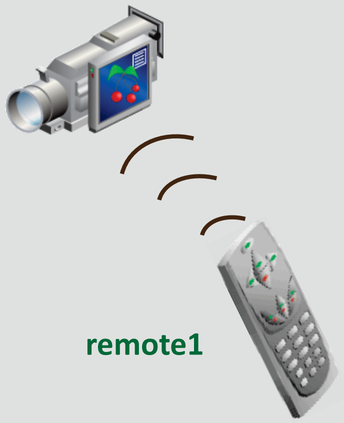
```
Camera remote1 = new Camera();
```

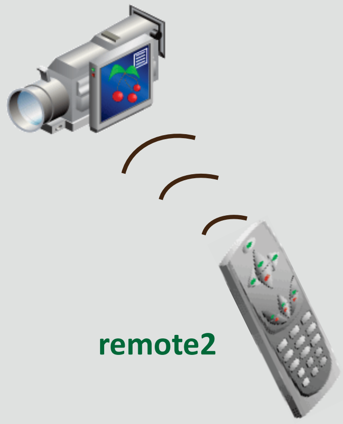


2. Call a method to have the Camera object do something

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

Working with Object References: Example 1

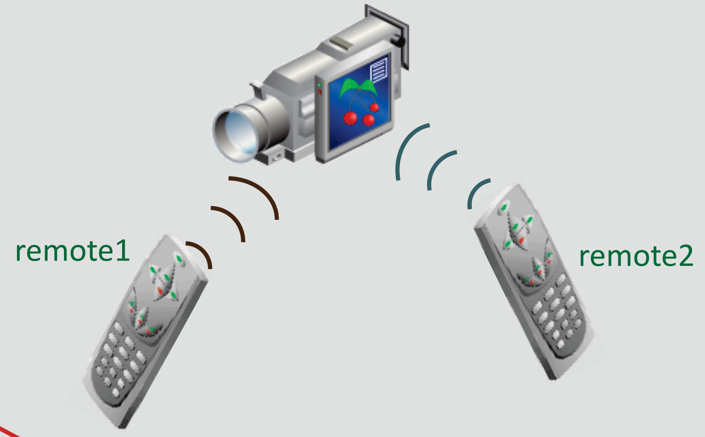




```
Camera remote1 = new Camera();
Camera remote2 = new Camera();

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

There are two Camera objects

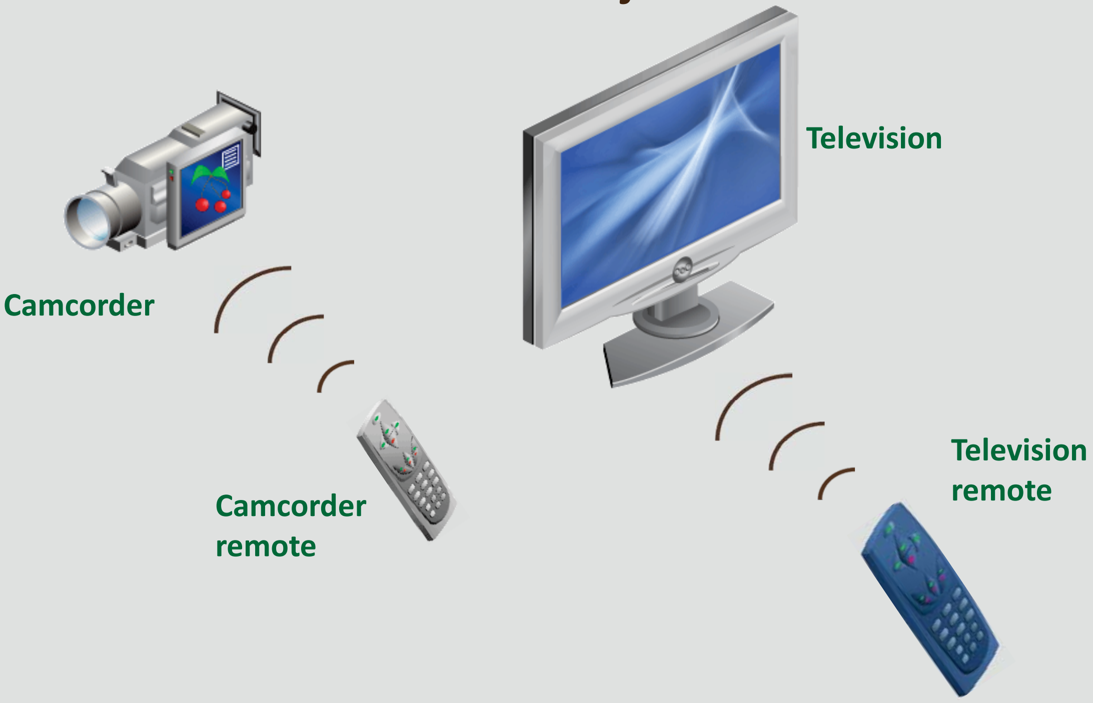
Working with Object References: Example 2



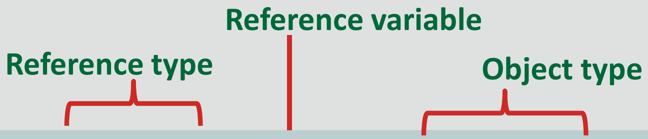
There’s only one Camera object

```
Camera remote1 = new Camera();

Camera remote2 = remote1;

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

References to Different Objects



References to Different Objects: Example


Reference variable

Reference type

Object type

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

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

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

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();
```

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
```

Stack Memory and Heap Memory

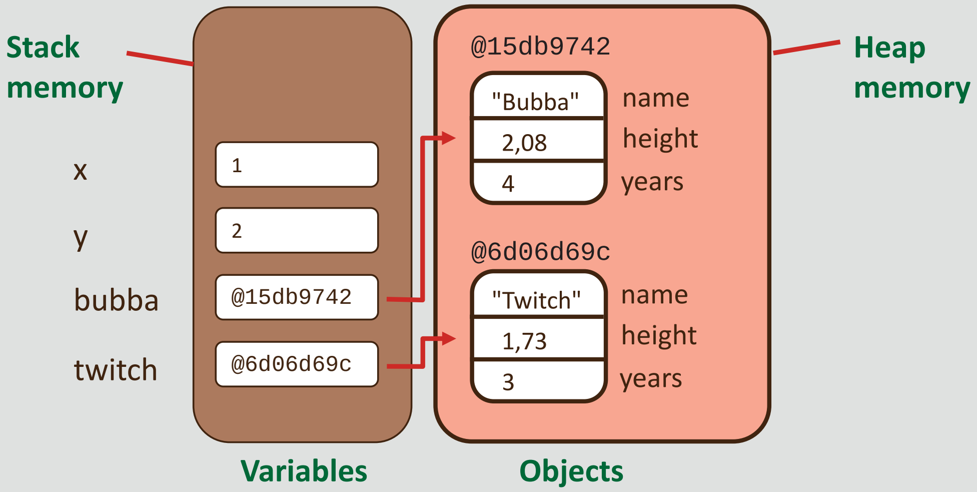
- Understanding the results of Exercise 2 requires an understanding of the types of memory that Java uses

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

- Heap memory is used to store …
  - Objects

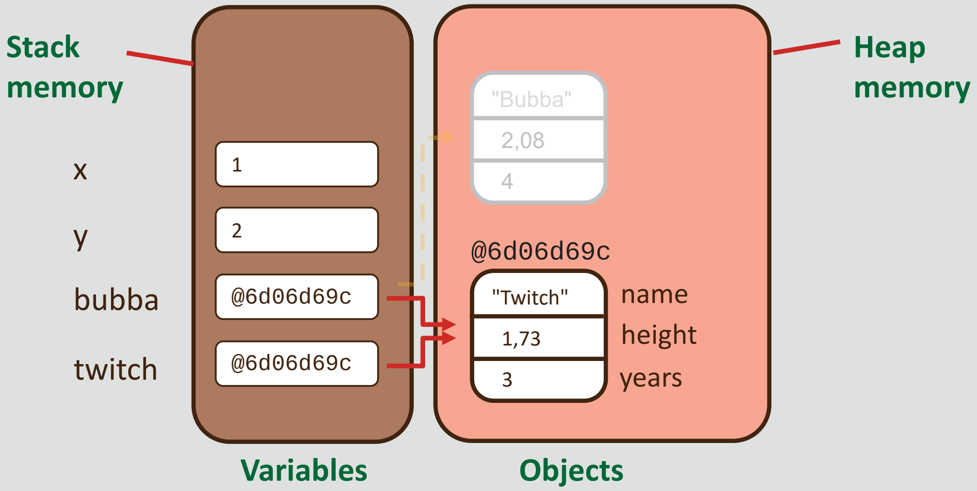
References and Objects in Memory

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



Assigning a Reference to Another Reference

```
bubba = twitch;
```



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
```

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
```

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:
```

```
Variable:     twitch,
              bubba
Name:         Twitch
Height:       5'8"
             (1,73m)
Sentence:     3 tahun
Memory Address: @6d06d69c
```

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";
```

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 ==

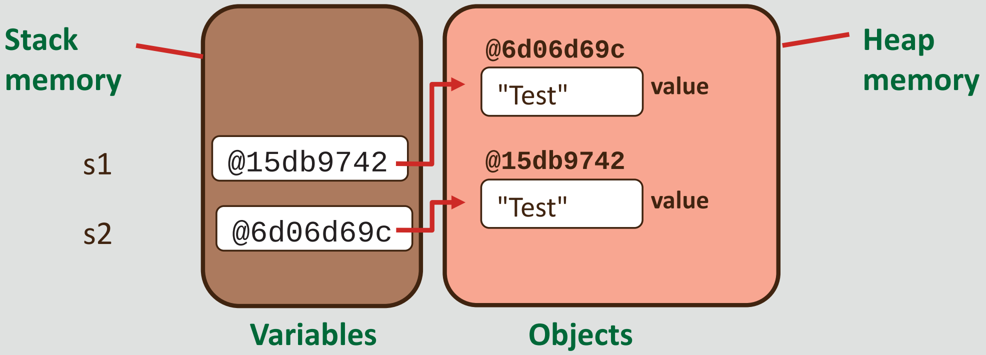
Variable:	bubba
Name:	Bubba
Height:	6'10" (2.08m)
Sentence:	4 years

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

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");
```



Stack memory

s1

s2

@15db9742

@6d06d69c

Variables

@6d06d69c

"Test" value

@15db9742

"Test" value

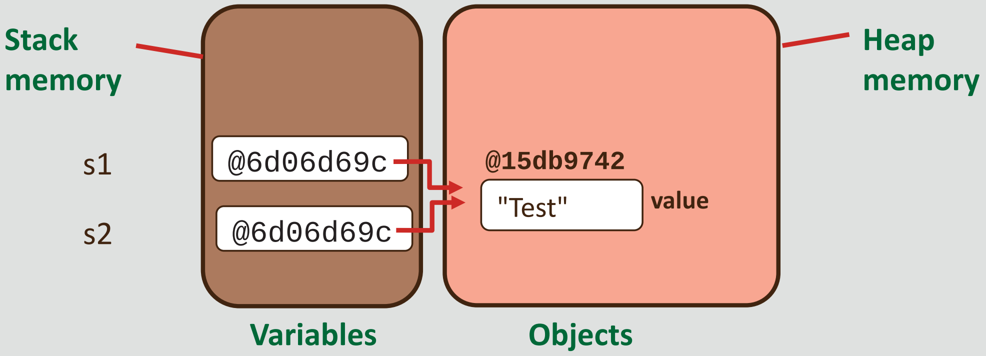
Objects

Heap memory

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

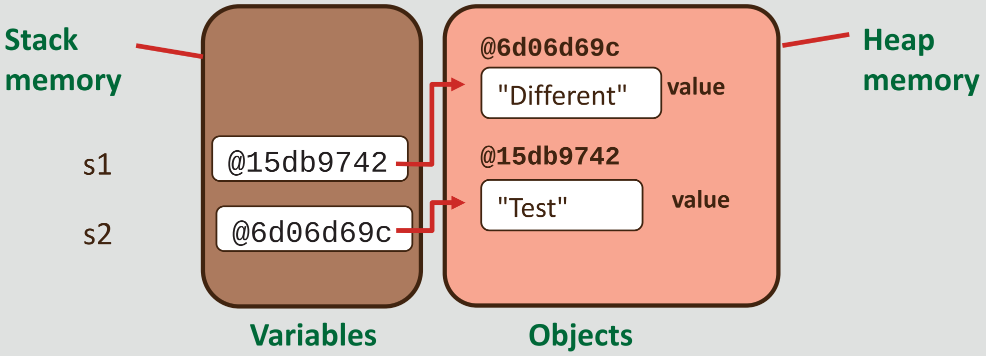
```
String s1 = "Test";
String s2 = "Test";
```



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";
```



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