References and Parameters

AP Computer Science

What is a reference?

References vs. Objects

- An object is the actual instance of the class stored in memory
- A reference describes the location in memory of a particular object
- A class is a blueprint for creating an object
- To actually create an instance of a class we use a constructor with the **new** keyword

Sample Student Class

- Here is an example of a Student class we will use for demonstration:

```
public class Student
{
   private String name;

   public Student()
   {
      name = "";
   }

   public Student(String n)
   {
      name = n;
   }
}
```

Default Constructor

Initialization Constructor

References vs. Objects

- Here is an example of a reference:

```
Student S1234;
```

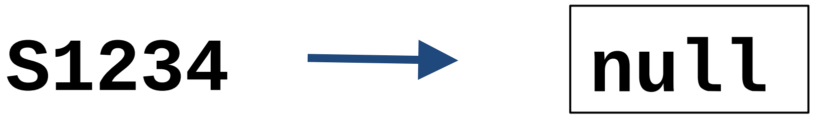
- At this point it does not store the location of an object
  - It points to a null location
- To create an object we need to instantiate it:

```
S1234 = new Student("Joe");
```

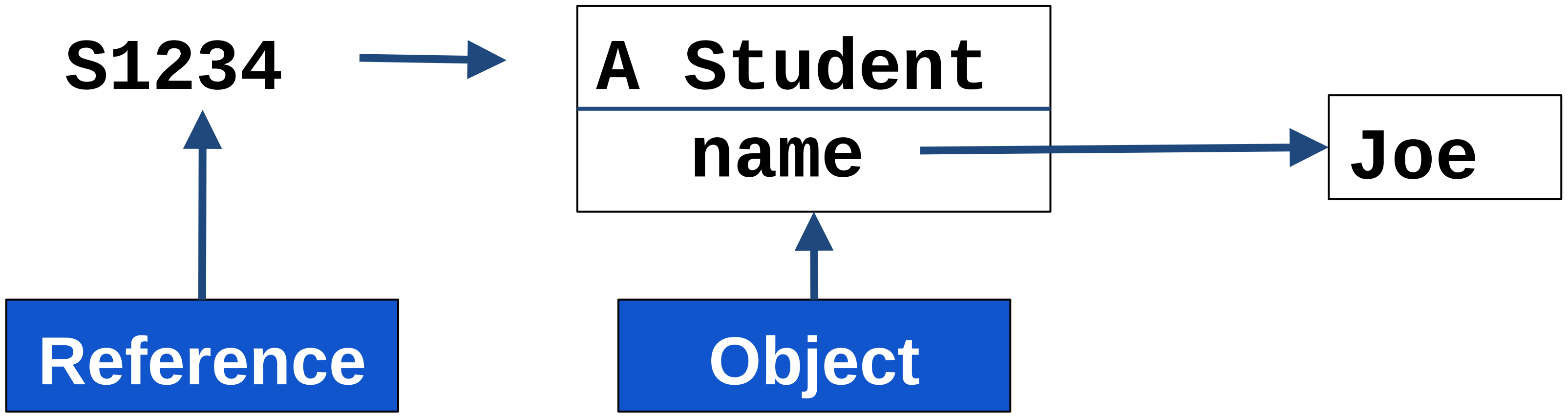
Reference **Constructor call**

References vs. Objects

```
Student S1234;
```



```
S1234 = new Student("Joe");
```



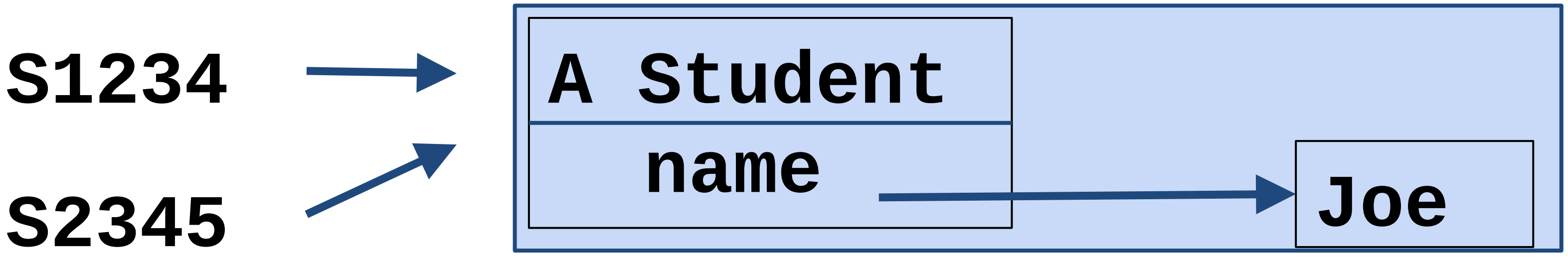
- What is name? What is Joe?

Aliasing

- Recall that there can be more than one reference to a given object
- Each reference is called an alias
- It is very important you understand the potential problems when there are multiple references to the same object

Aliasing with Objects

```
Student S1234 = new Student("Joe");
Student S2345 = S1234;
```



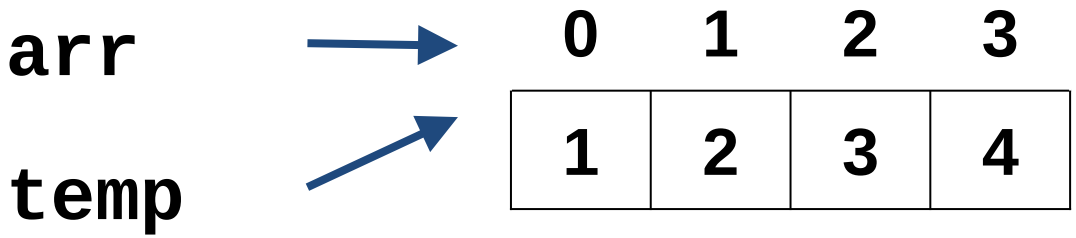
```
S1234.setName("Jane");
System.out.println(S1234.getName());
System.out.println(S2345.getName());
```

Output

```
Jane
Jane
```

Aliasing with Arrays

```
int[] arr = {1,2,3,4};
int[] temp = arr;
```



```
temp[2] = 8;
arr[1] = 7;
System.out.println(Arrays.toString(arr));
System.out.println(Arrays.toString(temp));
```

Output

```
[1,7,8,4]
[1,7,8,4]
```

Aliasing with Strings

```
String one = "Hello!";
String two = one;
System.out.println(one == two);
System.out.println(one.equals(two));
```

- In this case both references point to the same String
- This means == and equals will both be true

Output

```
true
true
```

Aliasing with Strings

```
String one = "Hello!";
String two = "Hello!";
System.out.println(one == two);
System.out.println(one.equals(two));
```

- In this case both references point to the same String
- This means == and equals will both be true

Output

```
true
true
```

Aliasing with Strings

```
String one = "Hello!";
String two = new String("Hello!");
System.out.println(one == two);
System.out.println(one.equals(two));
```

- In this case both references **do not** point to the same String

Output
false
true

What is a parameter?

Parameters

- A parameter is a value that is sent to a method when the method is called

```
public class Student
{
    private int age;
    public void setAge(int a)
    {
        age = a;
    }
}
```

- The parameter a is used by a caller to send a value to the method

Parameters

- Here is part of a main() method that uses the parameter to pass a value into the setName() method of a Student:

```
public class StudentRunner
{
   public static void main( String[] args )
   {
      Student S123 = new Student();
      S123.setAge(14);
   }
}
```

- 14 is passed to setAge() and becomes the value stored in a

Passing Parameters

Passing Primitive Variables

- Java passes all primitive parameters by VALUE

```
// code in main method
int age = 14;
S123.setAge(age);
```

- When this method call is placed a **copy** of the value of age is passed to setAge()
- At this point there is no connection between the value in the main method and the parameter in the method other than they have the same value

Passing Primitive Variables

- Java passes all primitive parameters by VALUE

```
// code in main method
int age = 14;
S123.setAge(age);
System.out.println(age);

// code in Student class
private int age;

public void setAge(int a)
{
      age = a;
      a = 10;
      System.out.println(a);
}
```

There is no relation between the age in the main and the age in the Student class

Output

```
10
14
```

Passing Reference Variables

- Java passes all reference parameters by VALUE
- However, this looks different with reference variables
- It passes a **copy** of the reference which is the location of the object
- This reference can be used to access the object and possibly change it

Passing Reference Variables

- Java passes all reference parameters by VALUE

```
// code in main method
String name = "Joe";
S123.setName(name);
System.out.println(name);

// code in Student class
private String name;

public void setName(String n)
{
      name = n;
      n = "Jane";
      System.out.println(n);
}
```

Name and n start by both pointing to a String "Joe", but the reassignment of n only changes n

Output

```
Jane
Joe
```

Passing Arrays

```
public class Temp{
     public void change(int[] temp){
          temp[0] = 5;
          temp[3] = 7;
     }
}

// code in main
int[] t = {1,2,3,4};
Temp obj = new Temp();
obj.change(t);
System.out.println(Arrays.toString(t));
```

Output

```
[5,2,3,7]
```

Passing Arrays

```
public class Temp{
      public void change(int[] temp){
            temp = new int[4];
            temp[0] = 5;
            temp[3] = 7;

      System.out.println(Arrays.toString(temp));
      }
}

// code in main
int[] t = {1,2,3,4};
Temp obj = new Temp();
obj.change(t);
System.out.println(Arrays.toString(t));
```

Output

```
[5,0,0,7]
[1,2,3,4]
```

Passing Objects

```
public class One{
      private String name;
      public void update()
            name = "Bob";
      }
      public String toString(){
            return name;
      }
}
public class Two{
      public void mys(One a, One b){
            a = b;
            b.update();
      }
}
// code in the main
Two test = new Two();
One x = new One("Jane");
One y = new One("Joe");
test.mys(x, y);
System.out.println(x + " " + y);
```

Output

```
Jane Bob
```

equals Method

Sample Student Class

- Here is an example of a Student class we will use for demonstration:

```
public class Student
{
    private String name;

    public Student()
    {
        name = "";
    }

    public Student(String n)
    {
        name = n;
    }
}
```

Default Constructor

Initialization Constructor

equals Method

```
Student S1234 = new Student("Joe");
Student S2345 = S1234;
Student S3456 = new Student("Joe");
System.out.println(S1234 == S2345);
System.out.println(S1234.equals(S2345));
System.out.println(S1234 == S3456);
System.out.println(S1234.equals(S3456));
```

- Why do we still get false on the last print statement?
- In the Student class we did not provide a way to check equality on two Student objects

Output

```
true
false
false
false
```

Updated Student Class

```
public class Student{
   private String name;
   public Student()
   {
      name = "";
   }
   public Student(String n)
   {
      name = n;
   }
   public boolean equals(Object obj)
   {
      Student s = (Student) obj;
      return name.equals(s.getName());
   }
}
```

Equals Method

equals Method

```java
Student S1234 = new Student("Joe");
Student S2345 = S1234;
Student S3456 = new Student("Joe");
System.out.println(S1234 == S2345);
System.out.println(S1234.equals(S2345));
System.out.println(S1234 == S3456);
System.out.println(S1234.equals(S3456));
```

- The equals method works as we intended now that the equals method has been written

Output

```
true
true
false
true
```

paramsworksheet3.doc