# Objects & Classes 2

CPSC 1181 – O.O.

Jeremy Hilliker

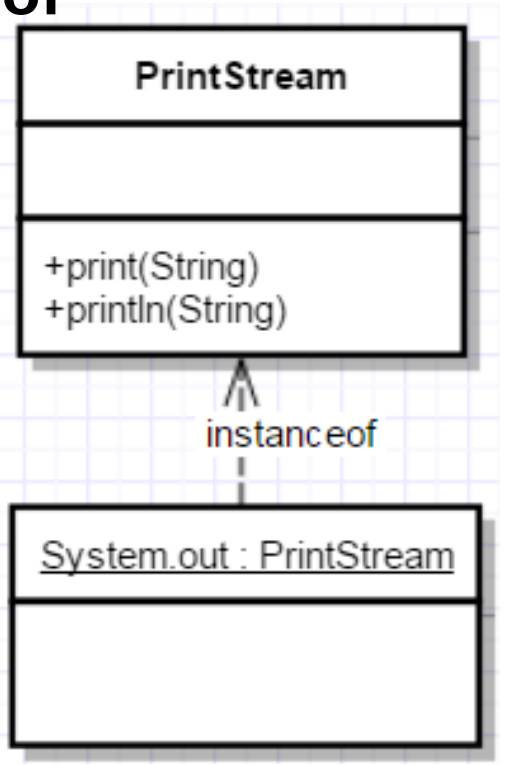
Summer 2017

Langara.
THE COLLEGE OF HIGHER LEARNING.

# Objectives

- Further understand classes and objects
- Use objects
  - Call methods
  - Construct objects
- Describe the attributes of a method

- Object
  - **Instance of a class**
  - Entity that you can manipulate
    - (identity, state, behaviours)
- Class
  - **Set of *objects* with the same set of *attributes* and *behaviours.***
  - Blueprint / contract
  - A type



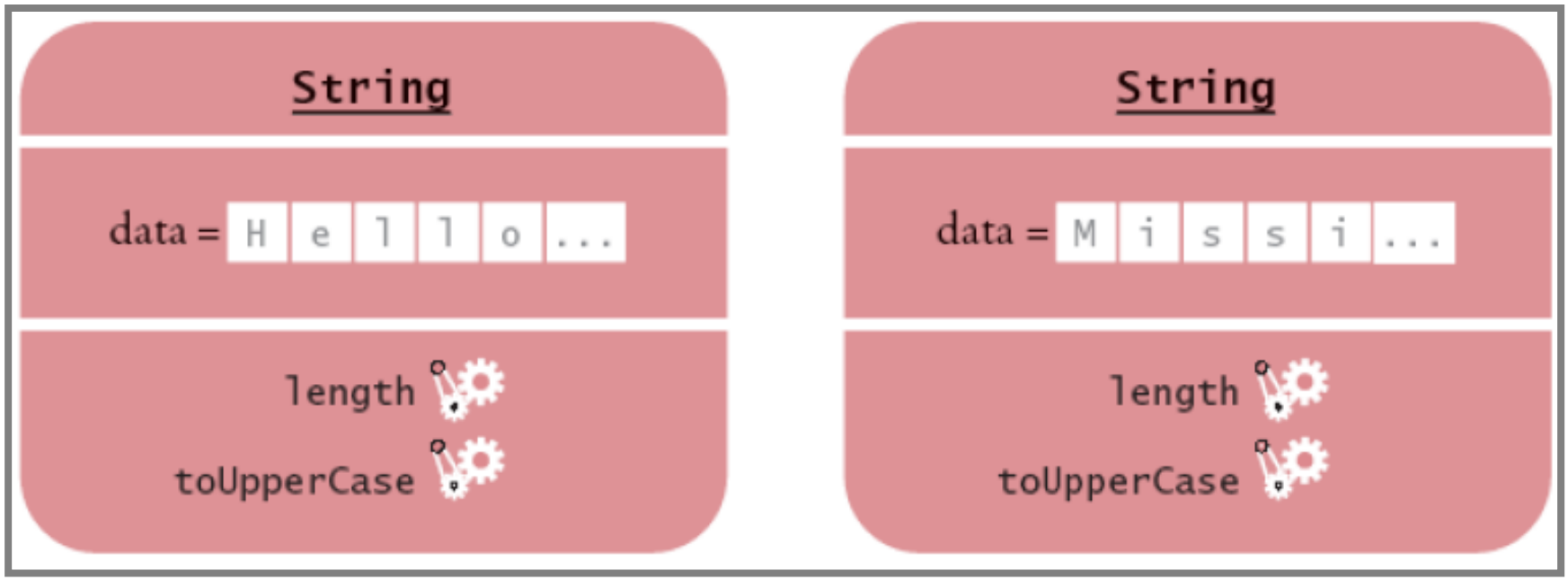
# Methods (operations, behaviours)

- **Method**: sequence of instructions to preform some task
  - (usually) access the data of the object
  - Manipulate an object by calling its methods
  - Accessors: get values from an object without changing state
  - Predicate: returns true / false
  - Mutators: set/change values in the object
- Class: "Set of *objects* with the same set of attributes and **behaviours**."
  - Class determines legal methods

```
String greeting = "Hello";
greeting.length();  // OK
greeting.println(); // Error
```

- Public interface: specifies what behaviours the class allows you to access on objects

# Strings




- int length():
  - the number of characters in the string
- String toUpperCase()
  - Creates a new string (from this one) that is all upper case

# Eg

```
String greeting = "Hello, World!";
int n = greeting.length(); // sets n to 13
```

```
String river = "Mississippi";
String bigRiver = river.toUpperCase();
            // sets bigRiver to "MISSISSIPPI"
```

```
System.out.length(); // This method call is an error
```

# Parameters: Explicit and Implicit

- Explicit Parameter:
  - Input to the method; passed by you

```
System.out.println(param);
param.length(); // no explicit param.
```

- Implicit Parameter:
  - the object on which the method is invoked

```
System.out.println(param);
```

  - “ this ”

- Bonus: OO and non-OO are duals

```
System.out.println(param);  // OO
println(System.out, param); // not OO
```

# Parameters: Formal and Actual

- Formal Parameters
  - The parameters of a method given by the class
    - Eg: Math.max(**int a, int b**)
    - int a, int b

- Actual Parameters
  - The values passed through those parameters
    - Eg: Math.max(**7, 12**)
    - 7, 12

# Return Values are Expressions

- A method that returns a value
  - Is something that has a value
  - So it is an expression

```
System.out.println(someString.length());
int n = greeting.length();
```

- Not all methods return a value

  “ void ”

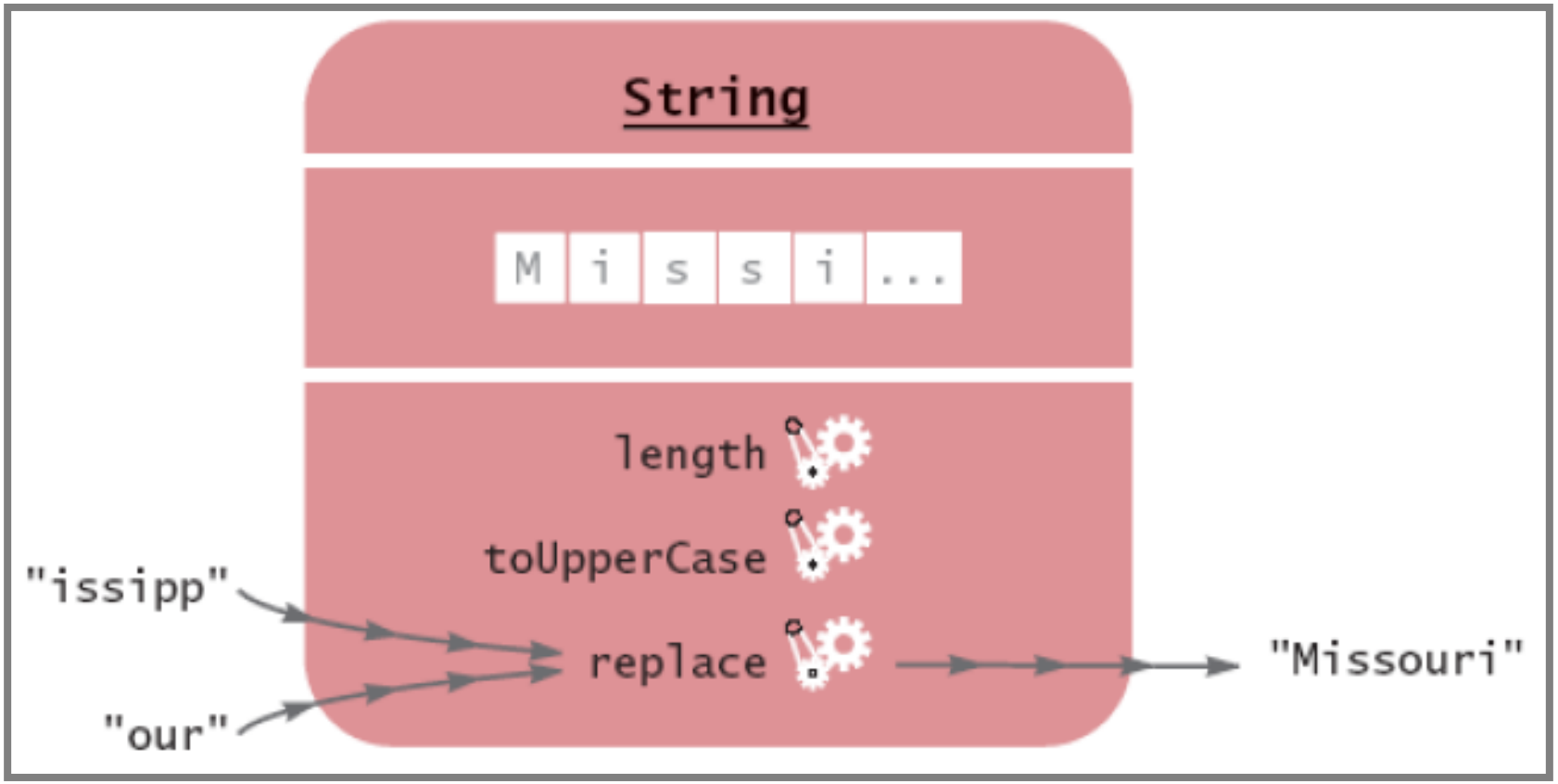
```
println(blah);
```

# Seeing it all

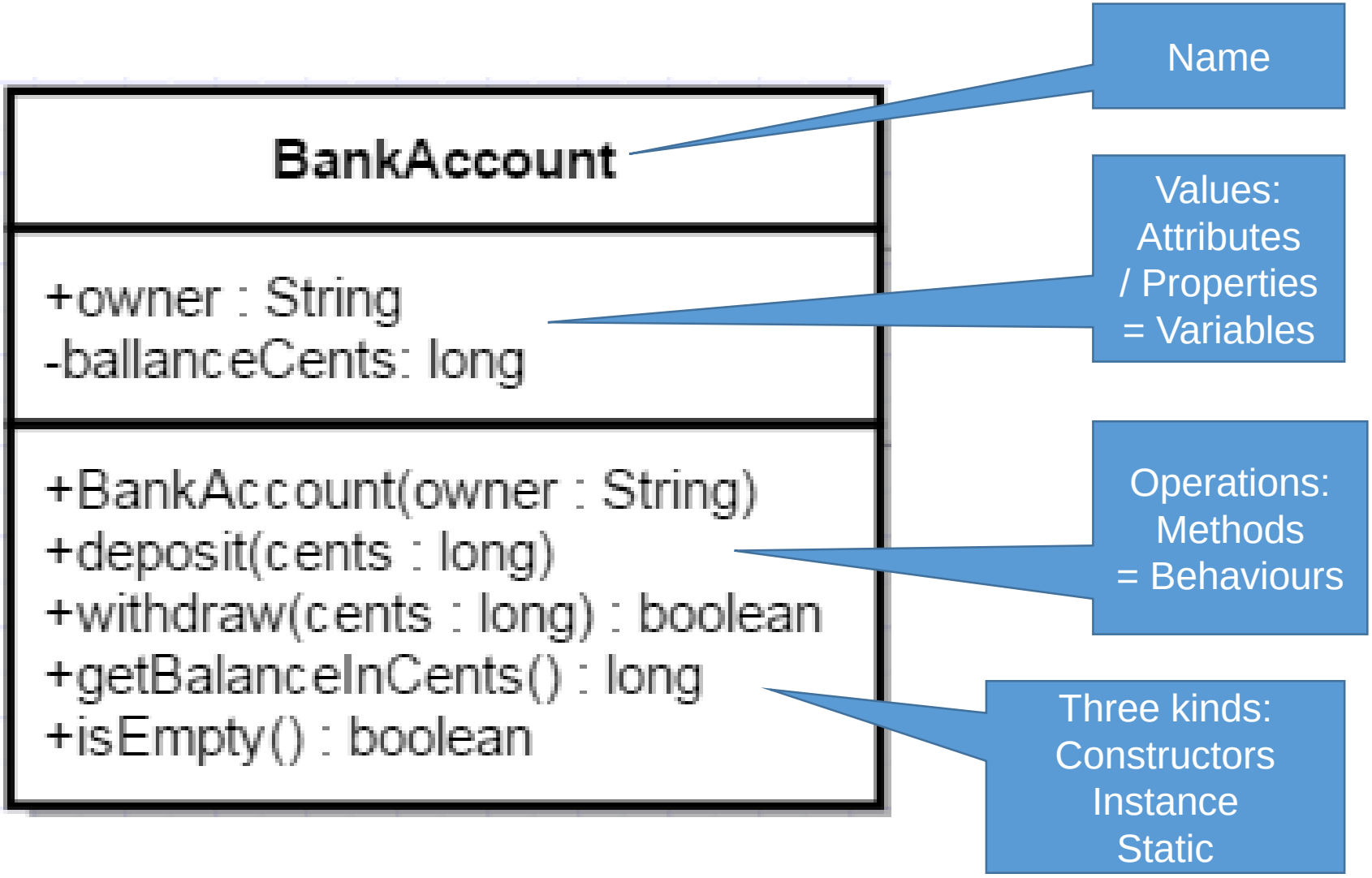
- String.replace()
  - “Replaces each substring of this string that matches the literal target sequence with the specified literal replacement sequence.” JSE 8 javadoc

```
String river = “Mississippi”;
State state = river.replace(“issipp”, “our”);
// state == “Missouri”;
```

  - 1 implicit param: “Missippi”
  - 2 explicit param: “issipp”, and “our”
  - A return value: <String> “Missouri”


String
M i s s i ...
length
toUpperCase
replace
"issipp"
"our"
"Missouri"

# Elements of a Class:



# Method Definitions

```
public String replace(
    String target,
    String replacement)
```

- Specifies part of the contract / interface of the class
- Note: implicit parameter defined by the enclosing class

- Modifiers
  - controls what objects can call the method
  - As well as some other keywords
- Return Type
  - the type returned, or void if none
- Method Name
  - same rules for identifiers
- Explicit, Formal Parameter list
  - in parenthesis, a comma-delimited list of pairs of types and names (or empty)
- Exception List
  - MAGIC! To be discussed later
- Method body
  - code enclosed between braces.

# Method Definitions

| Modifiers | Return Type | Method Name | Explicit Parameters | … |
|---|---|---|---|---|
| public | int | length | () | |
| public | String | replace | (String target, String replacement) | |
| | | | | |
| public static | int | main | (String[] args) | |
| | | | | |
| public | void | println | (String output) // PrintStreamWriter | |
| public | void | println | (int output) // PrintStreamWriter | |
| | | | | |

# Overloading

- A method is said to be ***overloaded*** if there are more than one method with:
  - the same name,
  - but a different parameter list (number or types)

```
public void println(String output)
public void println(int output)
```

# Accessors and Mutators

- Accessor:
  - Does not change state of the object

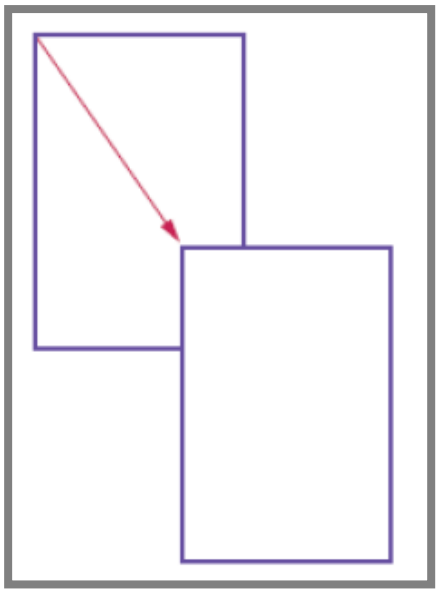
```
double width = box.getWidth();
```

- Predicate:
  - Determines is something is true

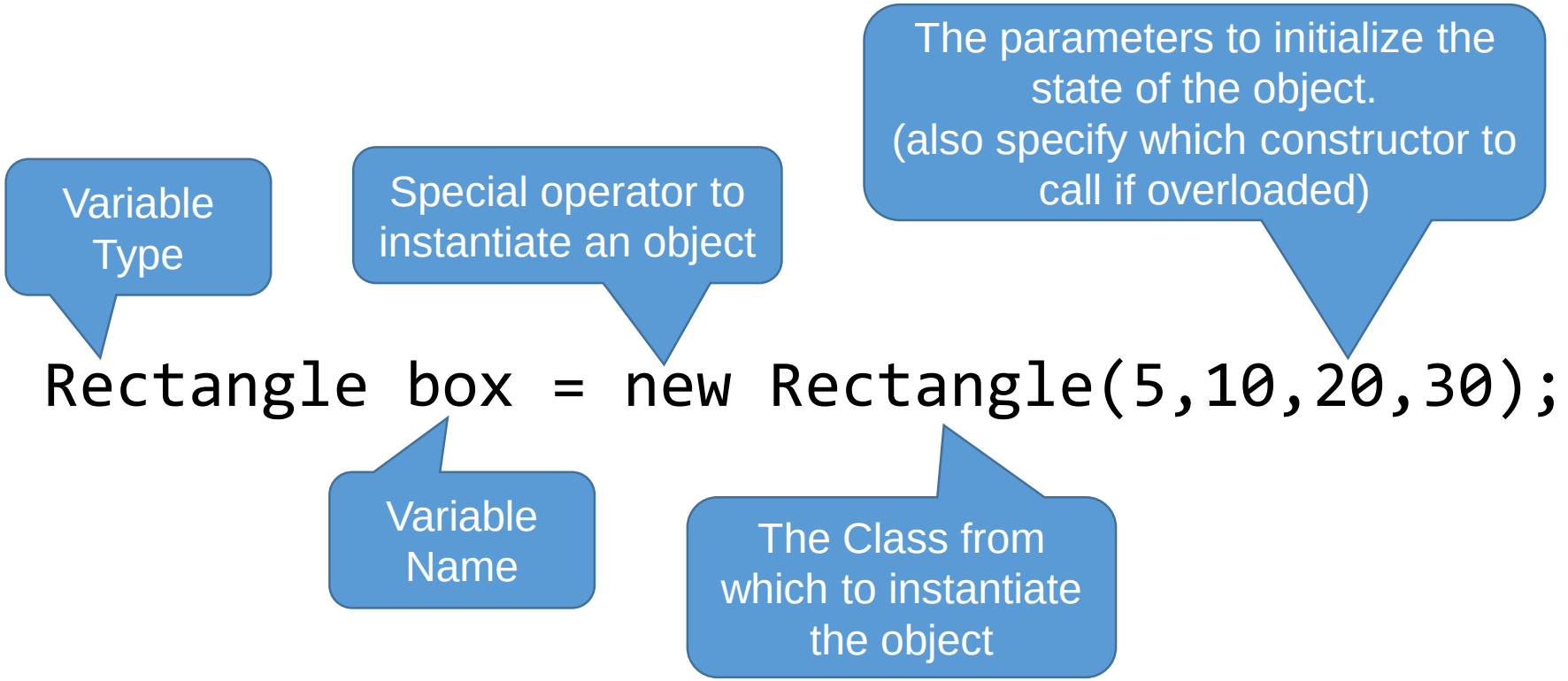
```
contains(r);
```

- Mutator:
  - Changes the state of the object

```
box.translate(15, 25);
```



# Constructing Objects

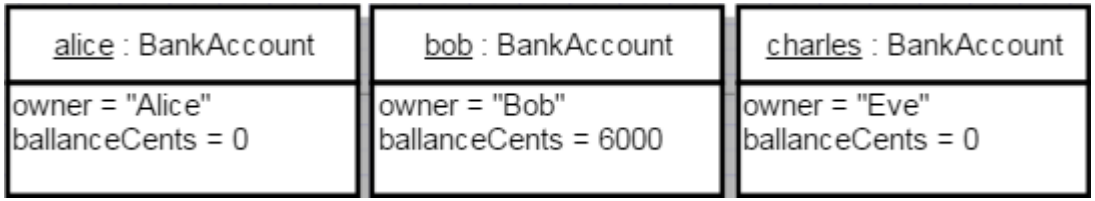


# Constructing Objects

- Creating a *new* object is called
  - ***Construction***, or ***instantiation***
- The method called to create the new object is called the method's ***constructor***
- The parameters passed to the constructor are called the ***construction parameters***
- The result of creating a new object is called an ***instance (of type …)***, or simply an ***object***.
- Some classes will have overloaded constructors
  - …

# Instantiate an Object of that Class

```
1   BankAccount alice = new BankAccount("Alice");
2   BankAccount bob = new BankAccount("Bob");
3   BankAccount charles = new BankAccount("Eve");
4
5   bob.deposit(6000);
6
```



# Overloading Constructors

- You can also overload constructors
- It's good practice to have your overloaded constructors call another constructor if possible

```
1  public class BankAccount {
2      private long cents;
3
4      public BankAccount() {
5          this(0);
6      }
7      public BankAccount(long aCents) {
8          if(aCents < 0) {
9              cents = 0;
10         } else {
11             cents = aCents;
12         }
13     }
```

# Recap

- Relationship between Class and Object
  - The class defines the object
- Calling methods
  - Return value as expression
  - Explicit & Implicit params
  - Formal & Actual params
- Overloading
- Constructors