Using Objects

- From last time: academic integrity
- What are objects?
  - a first example of an object
  - data abstraction and encapsulation
- classes and objects in Java
  - classes vs. objects
  - methods
  - constructing an object
  - mutators vs. accessors
  - object references
  - primitive values
  - Strings are special

Announcements

- The following students need to see me in o.h. today (or after class):
  - missed the first class
  - not officially enrolled in class (e.g., trying to get in) (even if you attended first class)
  - have no previous programming experience
- office hours through Mon posted on piazza (permanent office hour schedule coming soon)
- experience level survey coming soon (google form)
- Sample poll (that counts) available on piazza
- Zoom waiting room

Academic integrity policy

- See Course Syllabus for details.

Some reasons not to cheat

- Dishonest
- USC reputation
- Real world consequences
- Losing learning opportunity
- Academic consequences

Our first object…

What are classes/objects?

- just like functions are procedural abstractions…
  - give a name to an action
  - Examples:

What are classes/objects? (cont.)

- …classes are names for a data abstraction
  - encapsulates data + operations on that data (methods)
  - Examples: String, Watch, Car, Rectangle

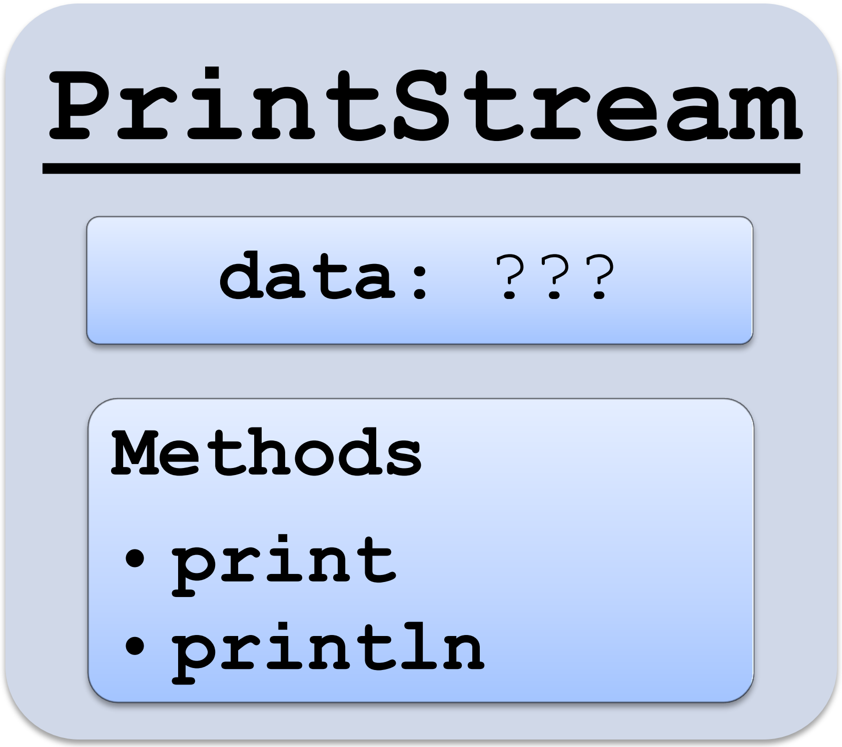
What are classes/objects? (cont.)

- with a class (data abstraction) **client** only knows…
  - name of class
  - what operations are
  - and how to use them

Class vs. Object

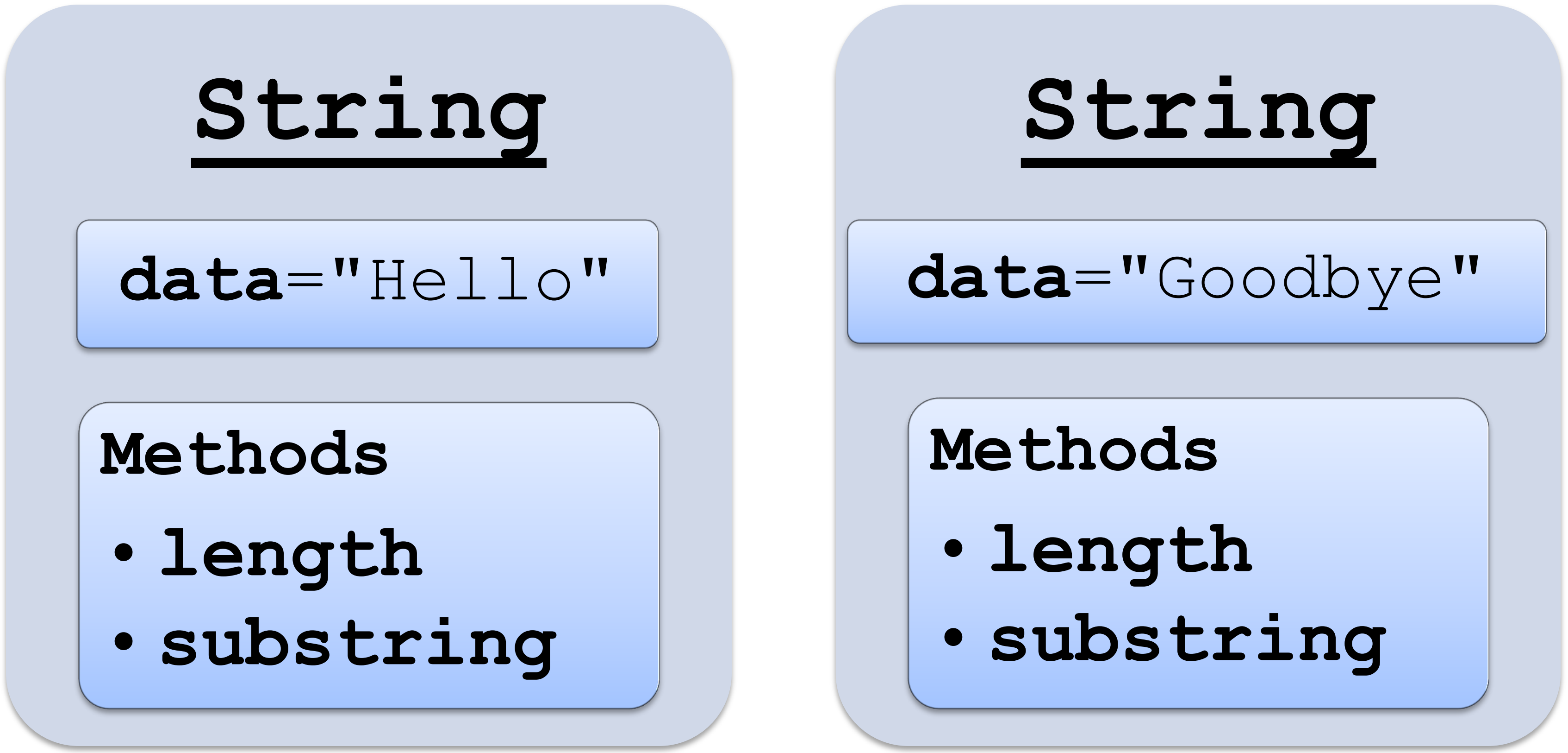
- Class is the type (aka, category)
  - all objects of a class have the same operations available
- Object is the instance of that type
- Can have multiple objects of the same class
- Ex:
  - class: Watch
  - object: my Timex watch

`System.out` object



```
System.out.println("Hello");
```

Multiple instances



```
String greeting = "Hello";
String lastWord= "Goodbye";
int n = lastWord.length();
```

Another String operation

- A convenient String operation (doesn't use the "dot" syntax):   +
- Appends two strings together
- Ex:

```
String summerFruit = "water" + "melon";
```

- Can use with String variables or String literals.

More with Strings

Use the **variables** below to print:

`Hello, Goodbye`

```
   String greeting = "Hello";
   String lastWord = "Goodbye";
```

Constructing objects

- Before we can call methods, have to create the object.
- The following statement does *not* create an object:

```
Rectangle rect;
```

- This does…

```
Rectangle rect = new Rectangle(5, 10, 20, 30);
```

  (parameters are: x, y, width, height)

- *constructor* call
- Can have multiple constrs. defined for a class. e.g.,

```
Rectangle r2 = new Rectangle();
```

Accessors and Mutators

- 2 kind of methods
- **accessors**: examine object
  - examples:

    ```
    greeting.length()
    rect.getWidth()
    ```
  - almost always have a return value
  - often use **`get`** in name

Mutators

- **mutators**: modify object
  - may or may not have a return value
  - changes internal state of object
  - sometimes use **`set`** in name
  - example…

Mutator example

Example: **`rect.translate(deltaX, deltaY)`**

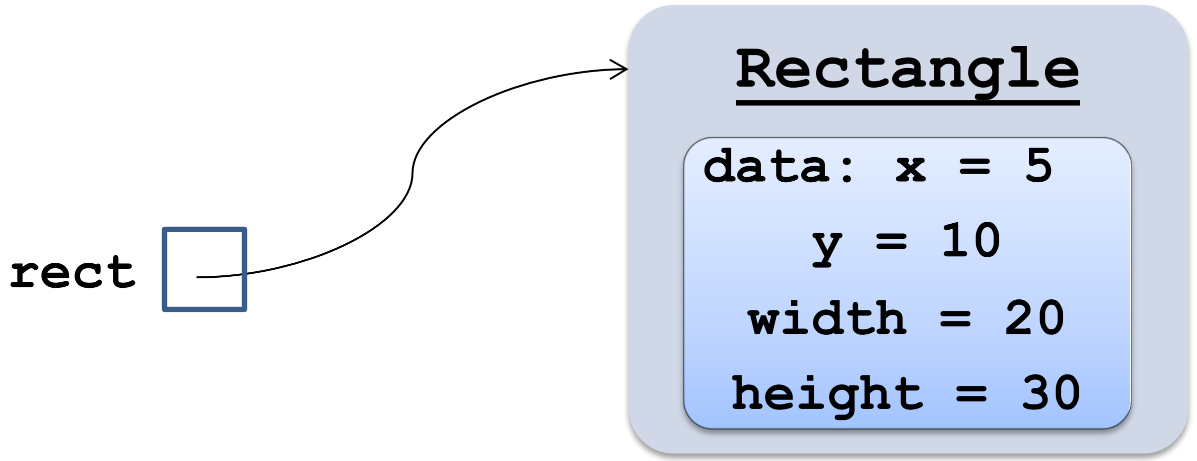
```
Rectangle rect = new Rectangle(5, 10, 20, 30);

rect.translate(10, 50);
```

Object references

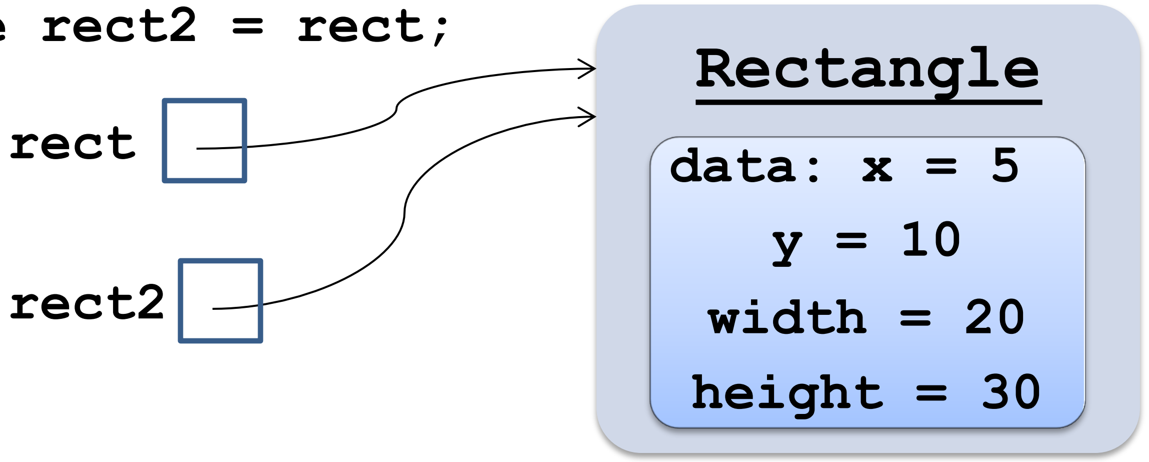
- variables of class types are not actually objects
- they are *object references*
- var contains the location of the object: *refers* to object
- e.g.

```
Rectangle rect = new Rectangle(5, 10, 20, 30);
```



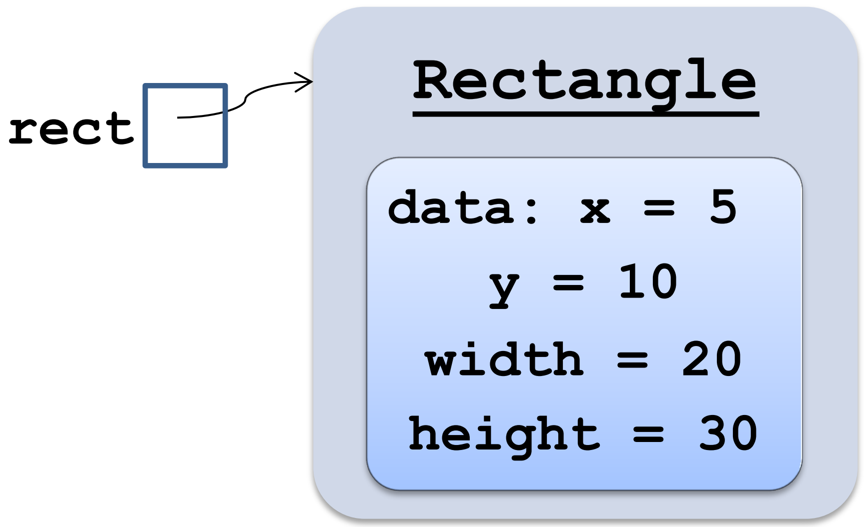
Two references to the same object

```
Rectangle rect = new Rectangle(5, 10, 20, 30);
Rectangle rect2 = rect;
```



Create two object instances with the same value

```
Rectangle rect = new Rectangle(5, 10, 20, 30);
Rectangle rect3;
```



Same object vs. same value

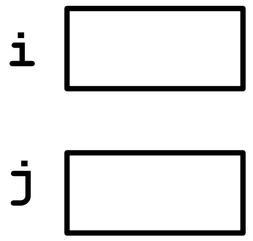
- Can compare objects for equality two ways:
  - Do they have the same value? (**`equals`** method)
  - example: **`a.equals(b)`**
  - Are they the same object? (**`==`**)
  - example: **`a == b`**

  -
    ```
    Rectangle rect = new Rectangle(5, 10, 15, 20);
    Rectangle rect2 = rect;
    Rectangle rect3 = new Rectangle(rect);
                                     // copy constructor
    ```

Objects vs. primitive values

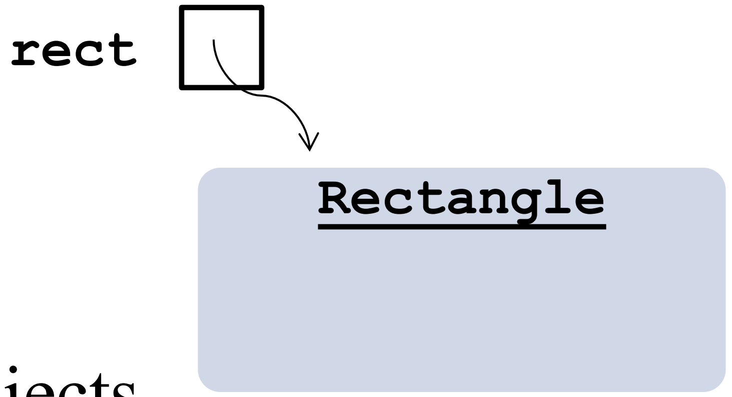
- 2 kinds of values: objects, primitive values
- primitive types: **`int, double, boolean`**, …

```
int i = 10;
int j = i;
j = 20;
```



- class types

```
Rectangle rect =
    new Rectangle(…);
rect2 = rect;
rect2.translate(…);
```



  – assignment does *not* copy objects

Objects vs. primitive values (cont.)

- primitive variables have **value semantics**
- object variables have **reference semantics**

`String` is special

- What about String?

```
String name = "Claire";
String name2 = name;
String name3 = "Claire";
```

- don't have to construct with new
- Strings may be shared internally for efficiency
- But it's safe: String is an *immutable* class

Immutable class

- Once object is created its state can never be changed.
- Has no mutators.

Another **`String`** method

`s.substring(startCharLoc, oneAfterEndCharLoc)`

- count chars starting from 0

```
String name = "Claire";
```

- *oneAfter - start = length of substring created*

```
String bearHouse = name.substring(
```

`substring` is not a mutator

```
String name = "Claire";
String name2 = name;
String bearHouse = name.substring(1,5);
name2 = name2.substring(2,5);
System.out.println(name + " " + name2);
```

(on Vocareum: item called *Lecture Code*, **`01-21`** dir: see Instructions on that item for how to access)

`String` is special: summary

- Strings instances are objects: have methods
- But can treat them more like numeric values:
  - don't need **`new`** to create one:

    ```
    String a = "foo";
    ```
  - assignment works as if it "copies" the String

    ```
    String b = a;
    ```
  - has "overloaded" Java + operator:

    ```
    a + b
    ```
  - can't change a String value once created:
    - **`charAt`**, **`replace`**, **`substring`**, etc. return new strings