Using Objects

• a first example of an object
• 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 after class or in o.h. today:
  – missed the first class
  – not officially enrolled in class (e.g., on waiting list, trying to get in) (even if you attended first class)
  – have no previous programming experience
Our first object...
What are classes/objects?

• just like functions are procedural abstractions
  – give a name to an action

• classes are names for data abstraction
  – encapsulates data + operations on that data (methods)

• client only knows…
  – name of class
  – what operations are
  – and how to use them
System.out object

PrintStream

data: ???

Methods
• print
• println

System.out.println("Hello");
Multiple instances

String

data="Hello"

Methods

• length
• substring

String

data="Goodbye"

Methods

• length
• substring

String greeting = "Hello";
String lastWord = "Goodbye";
int n = lastWord.length();
More with Strings

Use vars below to create: “Hello, Goodbye”

String greeting = "Hello";
String lastWord = "Goodbye";
Constructing objects

• Before we can call methods, have to create the object.
• The following 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. e.g.,

  rect = new Rectangle();
Accessors and Mutators

• 2 kind of methods
• accessors: examine object
  – examples:
    \texttt{hello.length()}, \texttt{rect.getWidth()}
  – almost always have a return value
  – often use \texttt{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)`

```java
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.

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

![Diagram showing object reference and data](image)
Two references to the same object

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

Rectangle

- data: x = 5
- y = 10
- width = 20
- height = 30
Create two object instances

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

- **Rectangle**
  - data: x = 5
  - y = 10
  - width = 20
  - height = 30
Objects vs. primitive values

• 2 kinds of values: objects, primitive values

• primitive types: `int`, `double`, `char`, ...

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

• class types

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

– assignment does not copy objects
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 it’s state can never be changed.
• Has no mutators.
Example `String` method

`s.substring(startCharLoc, oneAfterEndCharLoc)`

- count chars starting from 0
  
  ```java
  String name = "Claire";
  ```

- `oneAfter - start = length of substring created`
  
  ```java
  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);

Complete program in ~/csci455/code/01-14/StringEx.java
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:
    
    ```java
    String a = "foo";
    ```
  – assignment works as if it "copies" the String
    
    ```java
    String b = a;
    ```
  – has "overloaded" Java + operator:
    
    ```java
    a + b
    ```
  – can't change a String value once created:
    
    - `charAt`, `replace`, `substring`, etc. return new strings