

Intro to Programming II
Objects

Chris Brooks

Department of Computer Science
University of San Francisco

2-2: Objects

- Java is an *object-oriented* language.
- So what the heck is an object anyway?

2-3: Objects

- An object is a type of abstraction
- It provides a way of grouping together related data and functionality.
- Makes it easier to organize and extend your program.
- Also gives a “black box” effect.
 - Users of your objects don’t need to worry about how they work internally, just how to use them.

2-4: Classes and objects

- A class is a template or category
- An object is a particular instance of that class.
 - CartoonAnimals might be a class
 - BugsBunny, Tweety are instances of that class
- Classes let us specify behavior common to a set of objects.

2-5: An example

- Say we're making a drawing program, and we need to represent a collection of circles.
- A circle has an x and y coordinate for its center, plus a radius.
- We could do:

```
c1xval  = 3;  
c1yval  = 4;  
c1radius = 10;  
c2xval  = 5;  
c2yval  = 2;  
c2radius = 6;  
...
```

- (to be less messy, we could also store these in arrays)

2-6: An example

- This is not a good solution, though.
- Why not?

2-7: An example

- This is not a good solution, though.
- No grouping of a circle's components.
- Users need to know all about the internals of a circle.
- What if we don't know the number of circles in advance?

2-8: An example

- A better solution:

```
public class circle {  
    public int xval;  
    public int yval;  
    public int radius;  
}
```

- We have *encapsulated* the center and radius information inside the circle.

2-9: An example

- But we have two related variables representing the center.
- Perhaps we should group them as well.

```
public class point {
    public int xval;
    public int yval;
}
public class circle {
    public point center;
    public int radius;
}
```

2-10: Methods

- As we know, classes also contain methods.
- Methods are pieces of code that can be invoked on an object.
- They allow us to encapsulate both *state* and *behavior*.

2-11: Data hiding

- It's also important to protect instance data from outside users.
- One way to do this is by providing accessors and mutators
 - “setters and getters”
- Rather than the user modifying your object's data directly, they use a method to do it.
 - Reduces error
 - Hides implementation from the user.

2-12: Example

- For the following classes, add setter and getter methods for each of the instance variables.

```
public class point {
    public int xval;
    public int yval;
}
public class circle {
    public point center;
    public int radius;
}
```

2-13: Visibility modifiers

- *public* and *private* are used to indicate who can access a variable or method.
 - public: anyone can use it.
 - private: only available within that object.

2-14: Visibility modifiers

- Instance variables should be made private unless there's a compelling reason not to.
 - Use accessors and mutators to access and change data
- public methods are available for everyone to use.
- private methods can be used only within the object.
 - These are nice for “helper” methods that you don't want a user to call.

2-15: Example

- Design a book class.
- It should have instance variables for title, author, genre, publishers, copyright date. They should be strings.
- It should also have getters and setters for each of these.

2-16: Example

- Now create a Name class. It should have two members: `firstName` and `lastName`.
- Modify `Book` so that `author` is of type `Name`.
- Since you used setters and getters, users of your book class will never know!

2-17: Strings in Java

- Strings in Java are objects
- This mean that they have a set of methods they respond to:
 - `compareTo()`, `equals()`
 - `indexOf()`
 - `length()`
 - `replace()`
 - `startsWith()`, `endsWith()`
 - `etc`

2-18: Overloaded operators

- Unlike most other objects, Strings also have special behavior for creating *string literals* and for *concatenation*.
- String literals are strings where the value is known at compile time:
 - String s1 = "hello world"
 - String s2 = "USF"
 - String s3 = "I love Java"
- We can create a string without calling *new*.

2-19: Overloaded operators

- We can also use the '+' symbol to concatenate strings.
 - String s1 = "hello"
 - String s2 = "world"
 - String s3 = s1 + s2 // s3 = "hello world"
- This is a phenomenon called *overloading*; an operator is redefined to provide different functionality.

2-20: Example

- Now add two new methods to your Book class:
 - PrintSelf: it should concatenate the book's Title and Author and print them both out.
 - compareTo: this should take another book as an argument. It should return -1 if the other book comes before this book (ordered alphabetically by title), 0 if the titles are the same, and 1 if the other book comes after this one.
 - Use the String class' compareTo() method to do this.