

Objectives

- More control structures
- Object Oriented Programming
 - OOP review
 - Black-box programming
 - Creating classes in Java
 - State
 - Constructor
 - Methods

Sep 26, 2022

Sprenkle - CSCI209

1

1

Review

- What is the Java syntax for
 - Conditionals
 - Loops (two loops)
- How do you create a new array?
 - provide a few variations
- True or False: you can call methods on an array
- What are command-line arguments?
 - How do we access/use them in Java?

Sep 26, 2022

Sprenkle - CSCI209

2

2

MORE CONTROL STRUCTURES

Sep 26, 2022

Sprenkle - CSCI209

3

3

Control Flow: **while** Loops

● **while** loop

- Condition must be enclosed in parentheses
- Body of loop must be enclosed in `{}` if multiple statements

```
int counter = 0;
while (counter < 5) {
    System.out.println(counter);
    counter++; ← shortcut
}
System.out.println("Done: " + counter);
```

Sep 26, 2022

Sprenkle - CSCI209

Counter.java

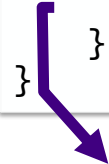
4

4

Changing control flow: **break**

- Exits the current loop

```
while ( <condition> ) {
    ...
    if( <something> ) { // now we're done!
        break;
    }
}
```



Sep 26, 2022

Sprenkle - CSCI209

5

5

Control Flow: **switch** statement

- Like a big **if/else if** statement
- Works with variables with datatypes **byte**, **short**, **char**, **int**, and **String**

```
int x = 3;
switch(x) {
    case 1:
        System.out.println("It's a 1.");
        break;
    case 2:
        System.out.println("It's a 2.");
        break;
    default:
        System.out.println("Not a 1 or 2.");
}
```

Sep 26, 2022

6

6

Control Flow: **switch** statement

```
switch(grade) {
    case 'a':
    case 'A':
        System.out.println("Congrats!");
        break;
    case 'b':
    case 'B':
        System.out.println("Not too shabby!");
        break;
    ... // Handle c, d, and f ...
    default:
        System.out.println("Error: not a grade");
}
```

Python now has match/case statement:
<https://docs.python.org/3/whatsnew/3.10.html>

Grades.java

7

7

Control Flow: foreach Loop

- Sun called “enhanced for” loop
- Iterate over all elements in an array (or Collection)
 - Similar to Python’s **for** loop

```
int[] a;
int result = 0;
. . .
for (int i : a) {
    result += i;
}
```

<https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html>

for each int element *i* in the array *a*,
 the loop body is executed

8

Summary: Python to Java Gotchas

- Every variable needs to be declared with its data type before it is used
- Scope of variables
- Need to use equals method to do more than just a “same object” check
- Syntax
 - Semicolons at the end of **statements**
 - Braces around blocks of code
 - Keywords
- Need to (1) compile and then (2) execute program

Sep 26, 2022

Sprenkle - CSCI209

9

9

Review: Classes & Objects

- **Classes** define template from which objects are made
 - “Cookie cutters”
 - Define **state** (aka fields or attributes)
 - Define **behavior**
- Many objects can be created of a class
 - Object: the cookie!
 - Ex: Many Mustangs created from Ford’s “blueprint”
 - Object is an **instance** of the class
- **Constructor**: a special method that constructs and initializes an object
 - After construction, can call methods on object

Sep 26, 2022

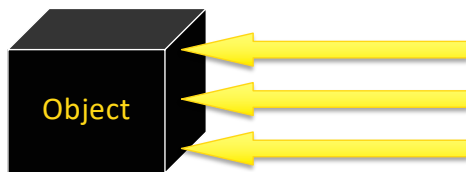
Sprenkle - CSCI209

10

10

Black-Box Programming

- **How** object does something doesn't matter
 - Example: if object *sorts*, does not matter to API user if implements merge or quick sort
- **What** object does matters (its **functionality**)
 - What object *exposes* to other objects
 - Referred to as “**black-box programming**” or **encapsulation**



- Has public **interface** that others can use
- Hides state from others

Sep 26, 2022

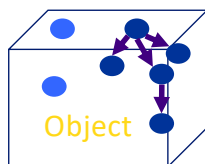
Sprenkle - CSCI209

11

11

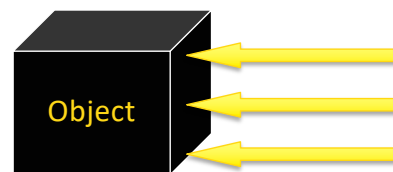
Discussion

What is the problem with white-box programming?



Others can see and manipulate object's internals

- May have unintended consequences



Java's structure helps us enforce black-box programming

Sep 26, 2022

Sprenkle - CSCI209

12

12

Access Modifiers

- A **public** method (or instance field) means that any object *of any class* can directly call the method (or access the field)
 - Least restrictive
- A **private** method (or instance field) means that any object *of the same class* can directly call this method (or access the field)
 - Most restrictive
- Additional access modifiers will be discussed with inheritance

In general, what access modifiers will we use for methods? For instance fields?

CREATING CLASSES

Classes and Objects

- Java is **pure object-oriented programming**
 - All data and methods in a program must be contained *within a class*
- But, for *data*, can use objects (e.g., Strings or Scanners) as well as primitive types (e.g., **int**, **double**, **char**)

Sep 26, 2022

Sprenkle - CSCI209

15

15

Example: Chicken class

- State
 - Name, weight, height
- Behavior
 - Accessor methods
 - **getWeight**, **getHeight**, **getName**
 - Convention: “get” for “getter” methods
 - Mutator methods
 - **feed**: adds weight and height when bird eats
 - **setName**



Sep 26, 2022

Sprenkle - CSCI209

16

16

General Java Class Structure

```
public class ClassName {
    // ----- INSTANCE VARIABLES -----
    // define variables that represent object's state
    private int inst_var;

    // ----- CONSTRUCTORS -----
    public ClassName() {
        // initialize data structures
    }

    // ----- METHODS -----
    public int getInfo() {
        return inst_var;
    }
}
```

Sep 26, 2022

Sprenkle - CSCI209

17

17

Example: Chicken class

- State
 - Name, weight, height
- Behavior
 - Accessor methods
 - getWeight, getHeight, getName
 - Convention: "get" for "getter" methods
 - Mutator methods
 - feed: adds weight, height
 - setName
 - Convention: "set" for "setter" methods

Discussion: what data types for instance variables?



Sep 26, 2022

Sprenkle - CSCI209

18

18

Instance Variables: Chicken.java

```
public class Chicken {
    // ----- INSTANCE VARIABLES -----
    private String name;
    private int height; // in cm
    private double weight; // in lbs
}
```

Instance variables are *declared*, with access modifier (**private**, in this case)

Constructor: Chicken.java

```
public class Chicken {
    // ----- INSTANCE VARIABLES -----
    private String name;
    private int height; // in cm
    private double weight; // in lbs

    // ----- CONSTRUCTORS -----
    public Chicken(String name, int h, double weight) {
        this.name = name;
        this.height = h;
        this.weight = weight;
    }
    ...
}
```

Observations? Thoughts? Guesses? Questions?

Constructor: Chicken.java

```

public class Chicken {
    // ----- INSTANCE VARIABLES -----
    private String name;
    private int height; // in cm
    // ----- CONSTRUCTORS -----
    public Chicken(String name, int h, double weight) {
        this.name = name;
        this.height = h;
        this.weight = weight;
    }
    ...
}

```

Constructor name same as class's name

Type and name for each parameter

Parameters don't need to be same names as instance var names

this: Special name for the constructed object, like **self** in Python (differentiate from parameters)

Sep 26, 2022

Sprenkle - CSCI209

21

21

Constructors

- **Constructor**: a special method that constructs and initializes an object
 - After construction, can call methods on object
- A constructor has the same name as its class
- Like `__init__` in Python

Sep 26, 2022

Sprenkle - CSCI209

22

22

Example: Chicken class

- State
 - Name, weight, height
- Behavior
 - Accessor methods
 - `getWeight`, `getHeight`, `getName`
 - Convention: “get” for “getter” methods
 - Mutator methods
 - `feed`: adds weight, height to this Chicken
 - `setName`



Discussion: What are the methods' **input** (parameters) and **output** (what is returned and its data type)?

Sep 26, 2022

Sprenkle - CSCI209

23

23

Methods: Chicken.java

```

... Type the method returns
// ----- Getter Methods -----
public String getName() {
    return this.name;
}

// ----- Mutator Methods -----
public void feed() {
    weight += .3;
    height += 1;
}
...
}

```

Chicken object's instance variables

Note that you don't have to use **this** when variables are unambiguous

Sep 26, 2022

Sprenkle - CSCI209

24

24

Note: Static vs Instance Methods

- `main` is a **static** method
- The methods we've been defining so far are *not* static
 - They do not have the **static** keyword as a modifier
 - They are therefore *instance* methods

More on this later...

Sep 26, 2022

Sprenkle - CSCI209

25

25

Constructing objects

- Given the `Chicken` **constructor**
`Chicken(String name, int height, double weight)`
create a chicken with the following characteristics
 - Name is "Fred", weight is 2.0, height is 38

```
Chicken chicken = new Chicken("Fred", 38, 2.0);
```

Sep 26, 2022

Sprenkle - CSCI209

26

26

Chicken's main method

- Where we'll do testing
 1. Create object
 2. Call methods
 3. Verify methods' results are what you expect
- When done testing, can move tests into separate test method
- Later: better ways to test

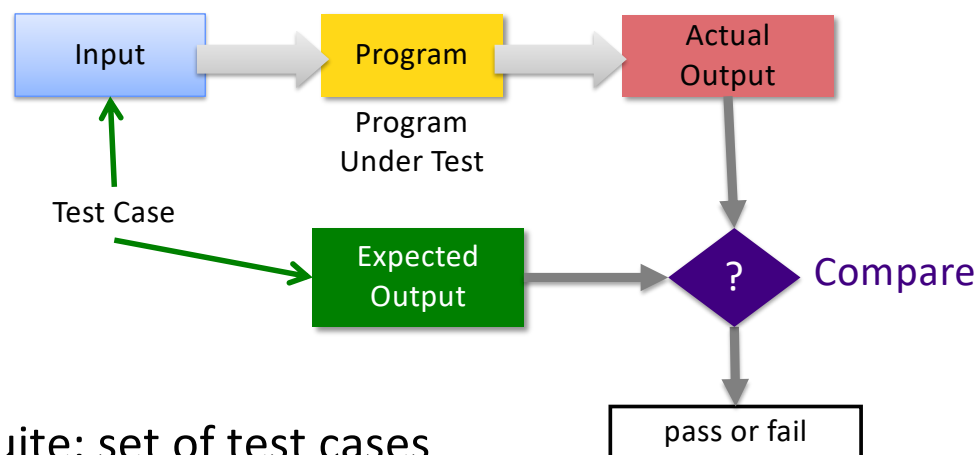
Sep 26, 2022

Sprenkle - CSCI209

27

27

Software Testing Process



- Test Suite: set of test cases

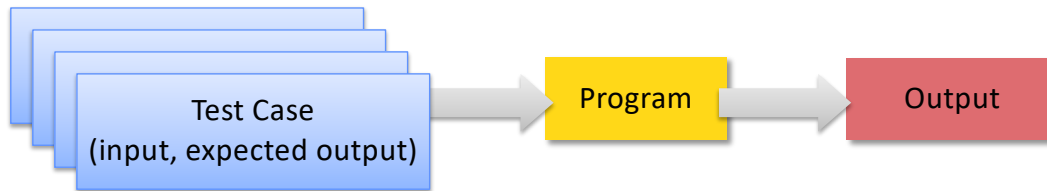
Oct 20, 2021

Sprenkle - CSCI209

28

28

Software Testing Process



- Tester plays devil's advocate
 - *Hopes* to reveal problems in the program using “good” test cases
 - Better tester finds than a customer!

How is **testing** different from **debugging**?

Oct 20, 2021

Sprenkle - CSCI209

29

29

Our Tests in main

- Execute the code to expose errors, automatically/programmatically
- Compare what we expect to the actual value
 - For mutators, this usually requires two steps:
 - Execute the method
 - Use other methods to check that the appropriate mutation occurred
- If actual and expected are different, displays an error message

Oct 3, 2022

Sprenkle - CSCI209

30

30

Class Development Process

1. Determine state
 - Declare state at top of class
2. Define constructor
 - Call constructor/create an object
3. Repeat
 - Write method or constructor
 - Test new method or constructor

Sep 26, 2022

Sprenkle - CSCI209

31

31

TODO

- Assignment 3 – due next Monday before class
 - OO programming
 - Recommendation
 - Parts 1 and 2 before Wednesday's class
 - Part 3 before Friday's class
 - Part 4 before Monday's class (most can be completed by Friday)
- Textbook – Read “Defining Classes in Java” up to but not including Inheritance

Sep 26, 2022

Sprenkle - CSCI209

32

32