

Objectives

- Static Fields and Methods
- Formatting

Review

- What is the format for documenting methods/constructors?
- What is *overloading*?
- How do we make an instance variable unchangeable after construction?
- How do we call a constructor within a constructor?
- What is the root of the Java class hierarchy?
- What is overriding?
 - What does the `@Override` annotation do?
- What is the method signature of the method we implement to allow pretty printing of objects we define?
- What is the method signature of the method we implement for determining if two objects are equivalent?
- What are benefits of programmatically/automatically testing (i.e., having the program execute test cases and determine if the test case fails)?

STATIC METHODS AND FIELDS

static Methods/Fields

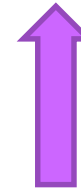
- For functionality/data that is specific to a *class*
 - And **not** specific to a particular object

static Methods/Fields Case Study: `java.lang.Math`

- To refer to static field: `ClassName.field`
 - Math's static fields: `PI`, `E`
 - Example: `Math.PI`
- To call static method: `ClassName.methodName(...)`
 - A static Math method: `static double sin(double a)`
 - `Math.sin( number );`
- No [public] constructor

Static Methods

ClassName.method(...)



- Do **not** operate on objects
 - i.e., you do **not** call object.staticMethod();
- Cannot access *instance* fields of their class
- Can access *static fields* of their class
 - Ex: Math class could have a *static* method that uses PI
- Similar to Python *functions* that are associated with the class

java.util.Arrays

- **Arrays** is a class in **java.util**
- Static methods for sorting, searching, `deepEquals`, fill arrays
- As seen in assignment 1: **`Arrays.sort(args)`**

Analyzing `java.lang.String` API

- Consider a “typical” ***instance*** (not static) method:

`String toUpperCase()`

➤ Converts all of the characters in *this String* to upper case

➤ Example use:

1) create a string `myString`

2) call `myString.toUpperCase()`

Analyzing java.lang.String API

- String toUpperCase()

- Converts all of the characters in *this String* to upper case
- Example: create a string, call myString.toUpperCase()

- static String valueOf(boolean b)

- Returns the string representation of the **boolean** argument
- Example use:
String.valueOf(**false**);

Why can/should
this method be **static**?

Discussion

Why is main static?

main Method

- Most common *static* method
- `main` does not get called on an object
 - Runs when a program starts
 - There are no objects yet but there is the class
- `main` executes and constructs the objects the program needs and will use
 - Like the *driver function* for the program

New rules for main

AltMainDemo.java
CompactSourceFileDemo.java

- Standardized in Java 25
- Can be static or instance
- **String[] args** parameter is optional
 - If don't need the command-line arguments
- Warning: Do not mix static and instance method with the same signature in one class
- Instance main: best for ***compact source files***

<https://docs.oracle.com/en/java/javase/25/language/compact-source-files-instance-main-methods.html>

Comparing Language Features

Compact Source Files

- New to Java 25+
- Less syntax
- No explicit class definition
- main looks like a function
- Lower barrier to entry for *early* programmers
- For “programming”

```
void main() {  
    System.out.println("Hello, world!");  
}
```

(Non-Compact?) Java Source Files

- All versions of Java
- Requires understanding of programming language concepts for **large-scale software development**

INTEGRATING STATIC INTO OUR CLASSES

Chicken static Field Examples

```
private static String farm = "MacDonald";  
private static final double WEIGHT_GAIN = .3;
```

What do the modifiers all mean for each variable?

Chicken objects

We have a bunch of Chicken objects

weight = 2.0
height = 38
name = "Fred"

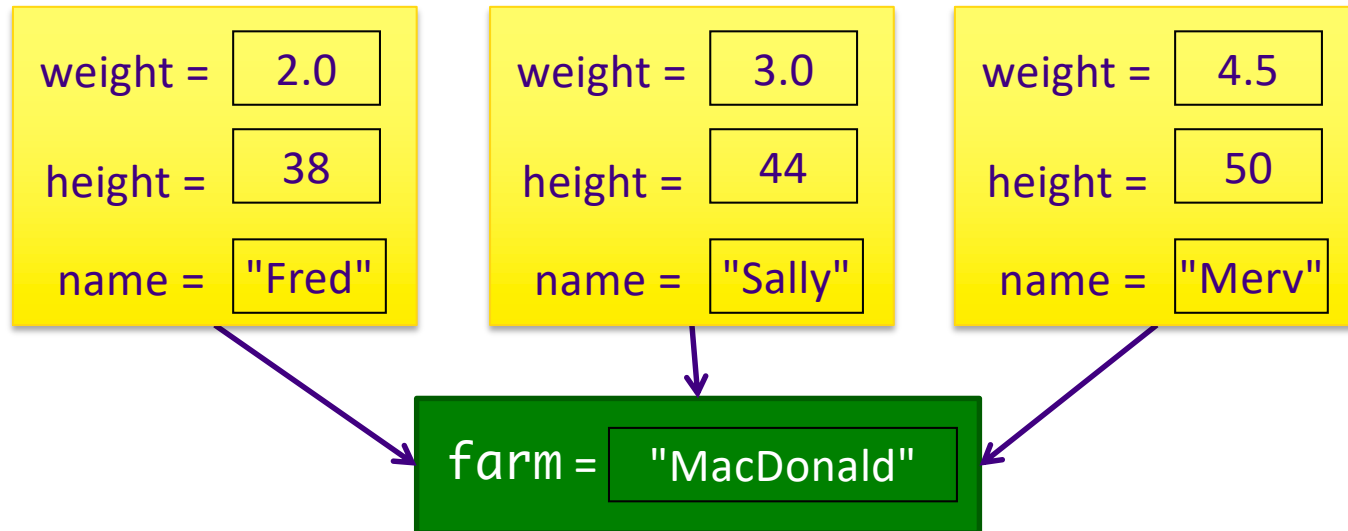
weight = 3.0
height = 44
name = "Sally"

weight = 4.5
height = 50
name = "Merv"

Chicken static Field Example

```
private static String farm = "MacDonald";
```

We have a bunch of Chicken objects



One variable shared by all members of the class.

Chicken static method Example

```
public static String getFarm() {  
    return farm;  
}
```

Call on Chicken class:

```
Chicken.getFarm();
```

More Examples

Benefit of WEIGHT_GAIN variable
(as opposed to .3)?

- Using static variable WEIGHT_GAIN:

```
public void feed() {  
    weight += WEIGHT_GAIN;  
    height += HEIGHT_GAIN;  
}
```

- Putting our testing code in a test method
- Creating a user interface for the chickens

Static Summary

- Static fields and methods are **part of a class** and **not** an object
 - Do not require an object of their class to be created to use them
- Static methods: similar role to a Python function
- When would we make a method *static*?
 - When a method does not need to access an *object's* state (fields) because all needed data are static fields in the class or are passed into the method
- When would we make a field *static*?
 - When it's a class variable (i.e., there should only be one for the whole class, shared among all objects of the class)

Summary: Class Design/Organization

- Fields

- Chosen first
- Placed at the beginning of class definition
- For class or instance?
- Have an access modifier, data type, variable name, and optional other modifiers (e.g., `final`)

- Constructors

- Have an access modifier
- Set all fields explicitly
- Use `this` keyword to access the object

- Methods

- For class or instance
- Have an access modifier
- Need to declare the return type
- Consider parameters

More static methods

FORMATTING

Formatting Strings: format

- Static method of the String class
- `String.format(<templatestring>, <value1>, <value2>, ..., <valuen>)`
Replacement values
- Semantics: creates, returns a **formatted string**
 - Means “format the templatestring, using the format(s) specified by **format specifiers** on the corresponding replacement values”
- Typically used with print statements

Formatting Strings

- **templatestring** is a template for the resulting string with format specifiers instead of the values

➤ For each format specifier in templatestring, should have a *replacement value*

```
String.format("%.2f", 3.14159 );
```

One format specifier

Corresponding replacement value

Returns "3.14"

Format Specifiers

[] mean optional

- General format:

%[flags][width][.precision]conversion

- conversion:

- Indicates the value's **type**/way to format

- width:

- *Minimum* number of character spaces reserved to display the entire value
 - Includes decimal point, digits before and after the decimal point and the sign

Conversion	Type
S	string
d	integer
f	float/double

Format Specifiers

- General format:

`%[flags][width][.precision]conversion`

- precision:

- Number of digits after the decimal point for **floating point** values

- flags:

- 0: zero fills
- +: adds a + sign before positive values
- -: left-justification (default is right-justification)

Example Formatted Strings

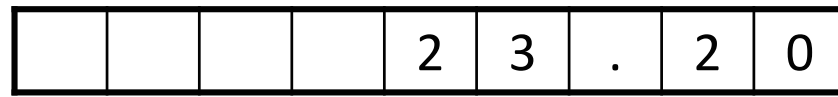
`String.format("%5d", 12)`



Field width is 5

" 12" Right-justified

`String.format("%9.2f", 23.1999)`



Precision is 2



Field width is 9

" 23.20"

- What if precision is bigger than the decimal places?
- What if field width is smaller than the length of the value?

Example Formatted Strings

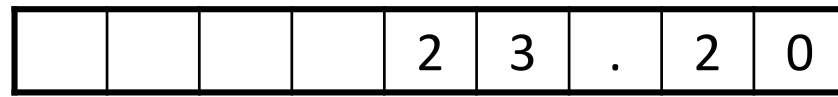
`String.format("%5d", 12)`



Field width is 5

" 12" Right-justified

`String.format("%9.2f", 23.1999)`



Precision is 2

Field width is 9

" 23.20"

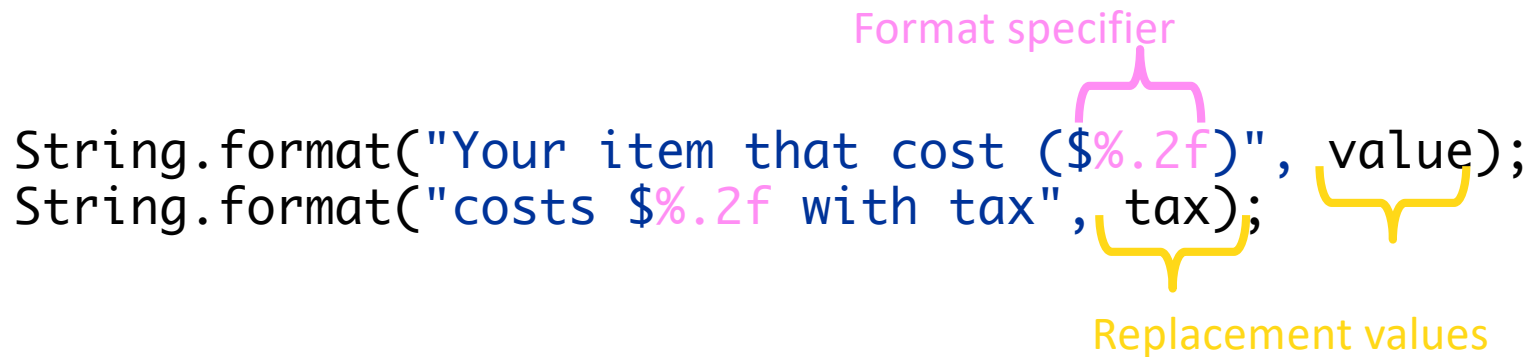
- What if precision is bigger than the decimal places?
 - Fills decimal with 0s
- What if field width is smaller than the length of the value?
 - String contains entire value

Partial Examples using format

String.format("Your item that cost (\$%.2f)", value);
String.format("costs \$%.2f with tax", tax);

Format specifier

Replacement values

The diagram highlights the components of the String.format method. In the first line, "Your item that cost (\$%.2f)", the "%.2f" is enclosed in a pink bracket labeled "Format specifier", and "value" is enclosed in a yellow bracket labeled "Replacement values". In the second line, "costs \$%.2f with tax", the "%.2f" is enclosed in a pink bracket labeled "Format specifier", and "tax" is enclosed in a yellow bracket labeled "Replacement values".

Common use case:

Save each of these in two (String) variables and print the Strings

Alternative: printf

- Similar to C

```
System.out.printf("Your item that cost ($%.2f)\n", value);  
System.out.printf("costs $%.2f with tax\n", tax);
```

Example: Printing Out Tables

- A table of temperature conversions

Temp F	Temp C	Temp K
-----	-----	-----
-459.7	-273.1	0.0
0.0	-17.8	255.4
32.0	0.0	273.1

- If we want to print data in rows, what is the template for what a row looks like?

➤ How do we make the column labels line up?

Example: Printing Out Tables

Using String.format

```
// example for one line of data in the table
double[] temps = {-459.7, -273.1, 0.0};

String tempFormat = "%10.1f %10.1f %10.1f";

System.out.println(String.format(tempFormat,
    temps[0], temps[1], temps[2]));
```

Example: Printing Out Tables

Using `String.format`

```
// example for one line of data in the table
double[] temps = {-459.7, -273.1, 0.0};

String tempFormat = "%10.1f %10.1f %10.1f";

System.out.println(String.format(tempFormat,
    temps[0], temps[1], temps[2]));
```

Example: Printing Out Tables

Using `System.out.printf`

```
// example for one line of data in the table
double[] temps = {-459.7, -273.1, 0.0};

String tempFormat = "%10.1f %10.1f %10.1f\n";

System.out.printf(tempFormat,
    temps[0], temps[1], temps[2]);
```

Static Method Practice

- Implement a **static** method called **average** that
 - Takes as parameters 2 integers
 - Returns the average of those 2 numbers
- What should the signature of this method look like?
 - Use the `main` method's signature to guide you
 - Discussion: what is the method's return type?
- Discussion: Why should this be a static method?
- Overload the method: takes two doubles

Putting it Together

- `main` will prompt user for two integers and display the result of getting the average
 - `main` is the driver
- Development process
 1. Hardcode the numbers
 2. Switch to user input
- How do we call the method?
 - Note that it is different because method is being called from another class

Assignment 2

- Lots of flexibility in design in Birthday and BirthdayParadox
- Lots of different correct designs
 - BUT many more incorrect or too-complicated designs
- Consider
 - If a variable should be a local variable, instance variable, or class variable
 - How can I break this into smaller problems? → Methods!
 - API for the methods: What is its input? What is its output (what is returned)?
- Test small parts!
- Use git well
 - When are good points to checkpoint (commit) or make a new branch?

Looking Ahead

- Assign 2 – due Wednesday 11:59 p.m.
- Exam 1 – Friday
 - Timed exam: 60 minutes, in class, hand-written
 - 1 letter-size page of notes for reference
 - Prep document online
 - 3 sections:
 - Very Short Answer, Short Answer, Coding