Announcements

● A Note on the Midterm
Announcements

- Lecture Feedback
Learning Goal for Today

Learn how to define our own variable types!
Plan for Today

- Review: Data Structures
- Classes
- Practice: Hedgehog Show
- Recap
Plan for Today

● Review: Data Structures
● Classes
● Practice: Hedgehog Show
● Recap
### Review: Data Structures

#### Arrays

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### 2D Arrays

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### ArrayLists

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

#### HashMaps

- **Key**: "dog"  
  **Value**: "woof"

- **Key**: "cat"  
  **Value**: "meow"
Fun fact: the collective noun for a group of hedgehogs is “an array of hedgehogs”.
Fun fact: the collective noun for a group of hedgehogs is “an array of hedgehogs”.

// An Aside
Plan for Today

- Review: Data Structures
- Classes
- Practice: Hedgehog Show
- Recap
Some Large Programs are in Java
Some Large Programs are in Java

How?
Defining New Variable Types

Inbox
Database

Email
Sender

Login
Manager

Email

User

Inbox
You’ve Been Using Variable Types

G0val → RandomGenerator → rgen → 22 → ArrayList

0 1 2 3
You’ve Been Using Variable Types

How can we make our own variable types?

GOval

RandomGenerator

ArrayList

?  →  rgen  →  22

0  1  2  3

/
A class defines a new variable type.
Why Is This Useful?

- A student registration system needs to store info about students, but Java has no `Student` variable type.

- A music synthesizer app might want to store information about different types of instruments, but Java has no `Instrument` variable type.

- An email program might have many emails that need to be stored, but Java has no `Email` type.
Why Is This Useful?

- However, Java does provide a feature for us to add new data types to the language: **classes**.
  - Writing a class defines a new variable type
- This lets you decompose your program across multiple files.
Why Is This Useful?

// We already have these variable types
GRect square = new GRect(100, 100);
String msg = "It’s almost August?!";
ArrayList<String> list = new ArrayList<String>();

// Using classes, we can now make these variable types!
Student s = new Student();
Email email = new Email();
Hedgehog walnoot = new Hedgehog("Walnoot");
Classes and Objects

- Every object is an instance of a class.
- Each new variable is a new instance.

- The class determines:
  - What state each instance has.
  - What behaviors each instance has.

- Each instance determines:
  - The specific values for that state information.
One instance of the GRect class

Another instance of the GRect class

rect1
rect2
**Classes Are Like Blueprints**

**class**: A template for a new type of variable.
# Classes Are Like Blueprints

<table>
<thead>
<tr>
<th>Hedgehog Class (blueprint)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>State:</strong></td>
</tr>
<tr>
<td>Has name</td>
</tr>
<tr>
<td>Has color</td>
</tr>
<tr>
<td>Has cuteness level</td>
</tr>
<tr>
<td><strong>Behavior:</strong></td>
</tr>
<tr>
<td>Can eat</td>
</tr>
<tr>
<td>Can run*</td>
</tr>
<tr>
<td>Can curl up</td>
</tr>
</tbody>
</table>
Classes Are Like Blueprints

Hedgehog Class (blueprint)

State:
- Has name
- Has color
- Has cuteness level

Behavior:
- Can eat
- Can run
- Can curl up

Hedgehog #1 (variable)

State: name = “Walnoot”
- color = Brown
- cuteness = 10 (Very cute)

Behavior:
- Can eat
- Can run
- Can curl up

Hedgehog #2 (variable)

State: name = “Nutmeg”
- color = Snowflake
- cuteness = 15 (VERY cute)

Behavior:
- Can eat
- Can run
- Can curl up

Hedgehog #3 (variable)

State: name = “Ruffles”
- color = Beige
- cuteness = 50 (speechless)

Behavior:
- Can eat
- Can run
- Can curl up
To design a new variable type, you must specify 3 things:

1. What subvariables make up this new variable type? (think: state)
2. How do you create a new variable of this type?
3. What methods can you call on a variable of this type? (think: behaviors)
What If...

What if we could write a program like this?

```java
BankAccount dukeAccount = new BankAccount("Duke", 50);
dukeAccount.deposit(50);
println("Duke now has: $" + dukeAccount.getBalance());

BankAccount karelAccount = new BankAccount("Karel");
karelAccount.deposit(50);
boolean success = karelAccount.withdraw(10);
if (success) {
    println("Karel withdrew $10.");
}
println(karelAccount);
```

Can I deposit beepers? Hmm, do they count as Bitcoin?
What If...

What if we could write a program like this?

```java
BankAccount dukeAccount = new BankAccount("Duke", 50);
dukeAccount.deposit(50);
println("Duke now has: 
" + dukeAccount.getBalance());

BankAccount karelAccount = new BankAccount("Karel");
karelAccount.deposit(50);
boolean success = karelAccount.withdraw(10);
if (success) {
    println("Karel withdrew $10.");
}
println(karelAccount);
```

Can I deposit beepers?  
Hmm, do they count as Bitcoin?
Creating a New Class

Let’s define a new variable type called `BankAccount` that represents information about a single person’s bank account.
Let’s define a new variable type called `BankAccount` that represents information about a single person’s bank account.

A `BankAccount`:
- Contains the name of account holder
- Contains the balance
Creating a New Class

Let’s define a new variable type called BankAccount that represents information about a single person’s bank account.

A BankAccount:

State
- Contains the name of account holder
- Contains the balance

Behavior
- Can deposit money
- Can withdraw money
public class <Classname> {

    // some awesome code

}
Creating a New Class

We're defining a thing called Classname

```java
public class <Classname> {
    // some awesome code
}
```
public class <Classname> extends <Superclass> {

    // some awesome code

}
We're defining a thing called Classname. Classname is a kind of Superclass.

```java
public class <Classname> extends <Superclass> {

    // some awesome code

}
```
Creating a New Class

public class BankAccount {

    // some awesome code

}
BankAccount.java

public class BankAccount {

    // some awesome code

}
public class BankAccount {

    // some awesome code

}
1. **What information is inside this variable type? (state)**
   - These are its private instance variables
public class BankAccount {
    // Step 1: the data inside a BankAccount
    private String name;
    private double balance;
}

Each BankAccount object has its own copy of all instance variables.
Creating a New Class

1. **What information is inside this variable type? (state)**
   - These are its private instance variables

2. **How do you create a new variable of this type?**
   - Constructor
Constructors

GRect rect = new GRect();

GRect square = new GRect(50, 50);

This is calling a special method!
The GRect constructor.
Constructors

BankAccount dukeAccount = new BankAccount("Duke", 50);
BankAccount karelAccount = new BankAccount("Karel");

The constructor is executed when a new object is created.
public class BankAccount {
    // Step 1: the data inside a BankAccount
    private String name;
    private double balance;

    // Step 2: how to create a new BankAccount
    public BankAccount(String accountName, double startBalance) {
        this.name = accountName;
        this.balance = startBalance;
    }
    public BankAccount(String accountName) {
        this.name = accountName;
        this.balance = 0;
    }
}
public class BankAccount {
    // Step 1: the data inside a BankAccount
    private String name;
    private double balance;

    // Step 2: how to create a new BankAccount
    public BankAccount(String accountName, double startBalance) {
        this.name = accountName;
        this.balance = startBalance;
    }

    public BankAccount(String accountName) {
        this.name = accountName;
        this.balance = 0;
    }
}
Constructors

- Initializes the state of new objects as they are created

  ```java
  public Classname(parameters) {
    statements;
  }
  ```

- The constructor runs when client calls `new Classname(...)`
- **No return type** specified: returns the new object being created
- If a class has no constructor, Java gives it a default constructor with no parameters; sets all fields to default values like 0 or `null`
Using Constructors

BankAccount dukeAccount = new BankAccount("Duke", 50);

dukeAccount

name = "Duke"
balance = 50

BankAccount(name, bal) {
    this.name = name;
    this.balance = bal;
}
Using Constructors

```java
BankAccount dukeAccount = new BankAccount("Duke", 50);
BankAccount karelAccount = new BankAccount("Karel");
```

**dukeAccount**
- name = "Duke"
- balance = 50

```java
BankAccount(name, bal) {
    this.name = name;
    this.balance = bal;
}
```

**karelAccount**
- name = "Karel"
- balance = 0

```java
BankAccount(name) {
    this.name = name;
    this.balance = 0;
}
```
Using Constructors

```java
BankAccount dukeAccount = new BankAccount("Duke", 50);
BankAccount karelAccount = new BankAccount("Karel");
```

When you call a constructor (with `new`):

1. Java creates a new “instance” of that class
2. The constructor initializes the object’s state (instance variables)
3. The newly created object is returned to your program
Creating a New Class

1. **What information is inside this variable type? (state)**
   - These are its private instance variables

2. **How do you create a new variable of this type?**
   - Constructor

3. **What can this new variable type do? (behaviors)**
   - These are its public methods
What if we could write a program like this?

```java
BankAccount dukeAccount = new BankAccount("Duke", 50);
dukeAccount.deposit(50);
println("Duke now has: "+ dukeAccount.getBalance());

BankAccount karelAccount = new BankAccount("Karel");
karelAccount.deposit(50);
boolean success = karelAccount.withdraw(10);
if (success) {
    println("Karel withdrew $10.");
}
println(karelAccount);
```

I really hope my beepers count as $$...
public class BankAccount {
    // Step 1: the data inside a BankAccount
    private String name;
    private double balance;

    // Step 2: how to create a new BankAccount (omitted)

    // Step 3: the things a BankAccount can do
    public void deposit(double amount) {
        this.balance += amount;
    }
    public boolean withdraw(double amount) {
        if (this.balance >= amount) {
            this.balance -= amount;
            return true;
        }
        return false;
    }
}
Methods defined in classes are called on an instance of that class.

```javascript
// Duke's Account
const dukeAccount = { 
  name: "Duke",
  balance: 50,
  deposit(amount) {
    this.balance += amount;
  }
};

// Karel's Account
const karelAccount = { 
  name: "Karel",
  balance: 0,
  deposit(amount) {
    this.balance += amount;
  }
};
```
Methods defined in classes are called on an instance of that class.

```javascript
dukeAccount.deposit(22);
```

```javascript
dukeAccount
name = "Duke"
balance = 50

deposit(amount) {
    this.balance += amount;
}
```

```javascript
karelAccount
name = "Karel"
balance = 0

deposit(amount) {
    this.balance += amount;
}
```
Methods defined in classes are called on an instance of that class.

```java
dukeAccount.deposit(22);
```

```java
dukeAccount
name = "Duke"
balance = 50
deposit(amount) {
    this.balance += amount;
}
karelAccount
name = "Karel"
balance = 0
deposit(amount) {
    this.balance += amount;
}
```
Methods defined in classes are called on an instance of that class.

```javascript
dukeAccount.deposit(22);
```

```javascript
name = “Duke”
balance = 72

deposit(amount) {
    this.balance += amount;
}
```

```javascript
name = “Karel”
balance = 0

deposit(amount) {
    this.balance += amount;
}
```
Methods defined in classes are called on an instance of that class.

```java
dukeAccount.deposit(22);
karelAccount.deposit(1.99);
```

```java
dukeAccount
name = "Duke"
balance = 72

deposit(amount) {
    this.balance += amount;
}
```

```java
karelAccount
name = "Karel"
balance = 0

deposit(amount) {
    this.balance += amount;
}
```
Methods defined in classes are called on an instance of that class.

```javascript
dukeAccount.deposit(22);
karelAccount.deposit(1.99);
```

```javascript
dukeAccount
name = “Duke”
balance = 72

deposit(amount) {
    this.balance += amount;
}
```

```javascript
karelAccount
name = “Karel”
balance = 0

deposit(amount) {
    this.balance += amount;
}
```
Defining Methods in Classes

Methods defined in classes are called on an instance of that class.

```javascript
dukeAccount.deposit(22);
karelAccount.deposit(1.99);
```

```javascript
dukeAccount
name = "Duke"
balance = 72

deposit(amount) {
    this.balance += amount;
}

karelAccount
name = "Karel"
balance = 1.99

deposit(amount) {
    this.balance += amount;
}
```
Defining Methods in Classes

Methods defined in classes are called on an instance of that class.

dukeAccount.deposit(22);
karelAccount.deposit(1.99);

dukeAccount
name = “Duke”
balance = 72

deposit(amount) {
    this.balance += amount;
}

karelAccount
name = “Karel”
balance = 1.99

deposit(amount) {
    this.balance += amount;
}
Wall of Abstraction

run

withdraw()

deposit()

Bank account data

Wall of Abstraction
private double balance;

- **encapsulation**: Hiding implementation details of an object from its clients.
  - Encapsulation provides *abstraction*. Separates external view (behavior) from internal view (state).
  - Encapsulation protects the integrity of an object’s data.

- A class’s instance variables should always be private.
  - No code outside the class can directly access/change it.
// BankAccountProgram.java
BankAccount dukeAccount = new BankAccount("Duke", 50);
dukeAccount.deposit(22);
println("Duke has "+ dukeAccount.balance); // doesn’t work

Only accessible inside
BankAccount.java
// BankAccountProgram.java
BankAccount dukeAccount = new BankAccount("Duke", 50);
dukeAccount.deposit(22);
println("Duke has $" + dukeAccount.getBalance()); // but this does!
Getters & Setters

- To allow the client to reference private instance variables, we define public methods in the class that
  - set an instance variable’s value (“getters”), and
  - get (return) an instance variable’s value (“setters”)

- Getters and setters prevent instance variables from being tampered with.
public class BankAccount {
    private String name;
    private double balance;
    ...
    // “setter”
    public void setName(String newName) {
        if (newName.length() > 0) {
            this.name = newName;
        }
    }
}
public class BankAccount {
    private String name;
    private double balance;
    ...

    // "setter"
    public void setName(String newName) {
        if (newName.length() > 0) {
            this.name = newName;
        }
    }

    // "getters"
    public String getName() {
        return this.name;
    }

    public double getBalance() {
        return this.balance;
    }
}
// BankAccountProgram.java
BankAccount dukeAccount = new BankAccount("Duke", 50);

dukeAccount.setName("Duke J.");
String name = dukeAccount.getName();
double balance = dukeAccount.getBalance();

println(name + " has $" + balance);  // “Duke J. has $50”
One Special Method...

How can we do this?

```java
BankAccount ba = new BankAccount(...);
println(ba); // ba isn’t a String!
```
How can we do this?

We define a `toString()` method

(inside the class file)

```java
BankAccount ba = new BankAccount(...);
println(ba);  // ba isn’t a String!
```

```java
public String toString() {
    return this.name
        + " has $" + this.balance;
}
```
One Special Method...

How can we do this?

We define a `toString()` method (inside the class file)

And now this works!

```java
BankAccount ba = new BankAccount(...);
println(ba); // ba isn’t a String!
```

```java
public String toString() {
    return this.name + " has $" + this.balance;
}
```

```java
BankAccount ba = new BankAccount(...);
println(ba); // prints “Duke has $50”
```
Classes Are Like Blueprints

BankAccount Class (blueprint)

State: Has name
Has balance

Behavior: Can deposit
Can withdraw

BankAccount #1 (variable)

State: name = “Duke”
balance = 50

Behavior: Can deposit
Can withdraw

BankAccount #1 (variable)

State: name = “Karel”
balance = 1.99

Behavior: Can deposit
Can withdraw
Making a Class ~ 3 Ingredients

1. Define the **variables** each instance stores (state)
2. Define the **constructor** used to make a new instance
3. Define the **methods** you can call on an instance (behaviors)

* all class methods and constructors have access to a **this** reference
Example: BankAccount
Plan for Today

- Review: Data Structures
- Classes
- Practice: Hedgehog Show
- Recap
True story: in a town in Washington, there is an annual Hedgehog Show! (ask Sarai)
Hedgehog Show

It’s a hedgehog day

A Hedgehog Gathering III will open at 10 a.m. today, Saturday, Oct. 8, and run until 5 p.m. at the Depot Arts Center and Gallery, 611 R Avenue in Anacortes. Admission is $5 for adults and $3 for seniors and children. Highlights of the show will be an discussion of hedgehog judging standards, as well as an auction and shopping for hedgehog merchandise. A portion of the show proceeds will benefit the Anacortes Community Theater, Depot Arts Center and Gallery, and Hedgehogs Northwest. Visit the Hedgehogs Northwest Web site at www.hhnw.hedgehogcentral.com.

The gathering is like a dog or cat show — but with hedgehogs instead. They are judged in three categories: attitude, health and shape.
Hedgehog Show

- Let’s help keep track of hedgehogs at the Hedgehog Show!
- To do that, we’ll need a new variable type, Hedgehog.
- How would you design a Hedgehog variable type?
Hedgehog Show

- Let’s help keep track of hedgehogs at the Hedgehog Show!
- To do that, we’ll need a new variable type, Hedgehog.
- How would you design a Hedgehog variable type?

The gathering is like a dog or cat show — but with hedgehogs instead. They are judged in three categories: attitude, health and shape.

- What state/properties (instance variables) and behaviors (methods) should it have?
Plan for Today

- Review: Data Structures
- Classes
- Practice: Hedgehog Show
- Recap
A class ...
A class defines a new variable type.
Making a Class ~ 3 Ingredients

1. Define the **variables** each instance stores (state)
2. Define the **constructor** used to make a new instance
3. Define the **methods** you can call on an instance (behaviors)

* all class methods and constructors have access to a **this** reference
Plan for Today

- Review: Data Structures
- Classes
- Practice: Hedgehog Show
- Recap

Next time: Classes Practice