CSCI 161
Introduction to Computer Science
Outline

- Abstraction and Modularity
  - Organism Class

- Useful APIs
  - String
  - Random

- Conclusion
How Would You Build an Organism?

- An Organism (modeled in software):
  - Can store and retrieve a thought
  - Can eat and retain food
  - Can digest food
  - Can sleep and wake up

- How much state do I need for this class!?
  - Lots of things to keep track!
    - Ingested food amount
    - Digested food amount
    - Awake or not?
    - What am I thinking about?
    - More!
How Would You Build This Lego City?

Would you build it bottom-up, and layer-by-layer?
Abstraction and Modularity!!

- Work on smaller pieces piecemeal, compose everything later
Another Example: **Building a car**

- Would we build it from the ground up?
- One expert knows how to create *everything* that goes into a car
- First, take raw aluminum ore and flatten/weld into a frame
- Second, make tires out of rubber, connect to tire rods (also crafted)
- Third build engine from scratch (every nut and bolt and belt, ...)
- and so on...
Abstraction and Modularity!!

- **Modularity**: Divide the whole into well-defined parts, which can be built separately and interact in well-defined ways.
  - In other words, "divide and conquer"
Abstraction and Modularity (Cont.)

- **Abstraction**: Ignore details of each part, focus on the high-level task

Lead Engineer:
I'm confident each subcomponent works, *even if I don't know how*. I just have to focus on making them work together!
Back to the Organism then...

- An Organism, like a car or lego city, is also complex
  - How can we modularize an organism? **What are its pieces?**
Toward Modularization

- Don't know its parts until we define what an Organism can do

- Let's say, all Organism objects can...
  - Can eat (and digest) food: `eat()`
  - Can store and retrieve a thought: `speak()`, `remember()`
  - Can sleep and wake up: `sleep()`, `wakeUp()`

Say... we have a Stomach class that lets us ingest and digest...

These are similar functions... Create a Brain class!
Let's Modularize an Organism

- First, let's remind ourselves what a Stomach can do
  - Hand out Stomach Application Programming Interface (API)

Organism Class
- eat()
- speak(), remember()
- sleep(), wakeUp()

Stomach Class (From Lab 3)

Brain Class (We'll write it next!)

We'll implement these after we write/test Brain,
Modularity: Define a Brain Class

- A brain can...
  - Hold a single *thought*, like "I'm hungry."
  - Keep track of whether it is *asleep*.

- **Ask:** What fields does the Brain need?

- **Ask:** What does a Brain know how to do?
  - `setThought` - Inputs a thought, and stores it in the brain.
  - `getThought` - Returns the current thought.
  - `setAwake` - Sets the status of the brain to either awake (true) or asleep (false)
  - `isAwake` - Returns whether or not the brain is awake
Stomach API

- An *application programming interface (API)* lists:
  - The available constructors
  - The available methods
- You should be able to look at an API and know everything how to use the class.

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>public Stomach()</td>
<td>Creates a new, empty Stomach</td>
</tr>
<tr>
<td>public int getAmountFood()</td>
<td>Returns the amount of food in the stomach</td>
</tr>
<tr>
<td>public int getAmountDigested()</td>
<td>Returns the amount of food digested</td>
</tr>
<tr>
<td>public void ingest(int amount)</td>
<td>Ingests the given amount of food. Ignores negative input.</td>
</tr>
<tr>
<td>public void digest()</td>
<td>Digests a random amount of food in the stomach. (Also removes that amount from stomach).</td>
</tr>
</tbody>
</table>
Here's the Brain's API

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Brain()</code></td>
<td>Creates a new, empty Brain</td>
</tr>
<tr>
<td><code>public void setThought(String newThought)</code></td>
<td>Remembers the specified thought.</td>
</tr>
<tr>
<td><code>public String getThought()</code></td>
<td>Returns the current thought.</td>
</tr>
<tr>
<td><code>public void setAwake(boolean newStatus)</code></td>
<td>Sets the status of the brain to either awake (true) or asleep (false)</td>
</tr>
<tr>
<td><code>public void isAwake()</code></td>
<td>Returns whether the brain is awake (true) or asleep (false)</td>
</tr>
</tbody>
</table>
Now we can write Organism class without writing brain and stomach functions. Just *use* them!

Stomach Class (From Lab 3)

Brain Class (We just wrote it)

Organism Class (Now we can write it!)
Abstraction: Now Write the Organism Class!

- An organism has the following fields:
  - A name, a stomach, and a brain

- An organism's methods:
  - **sleep** - Prints "Zzz" and puts brain to sleep (no action if already sleeping)
  - **wakeup** - Prints "Yawn" and wakes brain up
    - No action if not sleeping
  - **eat** - Inputs an amount to ingest, prints "Nom nom" to screen, digests too.
    - No action if sleeping
  - **speak** - Prints the current thought
    - No action if sleeping
  - **remember** - Inputs a thought and remembers it in the brain. Prints "Interesting..." to the screen.
    - No action if sleeping
Outline

- **Data Types**
  - Primitives vs. Classes

- **Abstraction and Modularity**
  - Organism Class

- **Useful APIs**
  - String
  - Random

- **Conclusion**
Java's Random Class

- Saw this in Guessing Game Lab
  - Need to `import` the class first

- Learn how to use it just by reading its `application programming interface (API)`
  - [https://docs.oracle.com/javase/7/docs/api/java/util/Random.html](https://docs.oracle.com/javase/7/docs/api/java/util/Random.html)

---

**Signature**

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>public Random()</code></td>
<td>Creates a new random number generator</td>
</tr>
<tr>
<td><code>public boolean nextBoolean()</code></td>
<td>Returns a random boolean value (50% probability)</td>
</tr>
<tr>
<td><code>public double nextDouble()</code></td>
<td>Returns a random value between 0.0 (incl) and 1.0 (excl)</td>
</tr>
<tr>
<td><code>public int nextInt()</code></td>
<td>Returns a random integer between 0 (incl) and 2^32 (excl)</td>
</tr>
<tr>
<td><code>public int nextInt(int n)</code></td>
<td>Returns a random integer between 0 (incl) to n (excl)</td>
</tr>
</tbody>
</table>
# String API (Selected Methods)

<table>
<thead>
<tr>
<th>Signature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>public boolean equals(String other)</strong></td>
<td>Checks content equality with given String</td>
</tr>
<tr>
<td><strong>public boolean equalsIgnoreCase(String other)</strong></td>
<td>Same as above, but case-insensitive</td>
</tr>
<tr>
<td><strong>public int indexOf(String str)</strong></td>
<td>Returns starting position of str if found, or -1 if not found</td>
</tr>
<tr>
<td><strong>public int length()</strong></td>
<td>Gets the length of the String</td>
</tr>
<tr>
<td><strong>public String replace(String target, String rep)</strong></td>
<td>Returns this String resulting from replacing target with rep</td>
</tr>
<tr>
<td><strong>public String substring(int begin, int end)</strong></td>
<td>Returns this String starting from position begin, ending at position end-1</td>
</tr>
<tr>
<td><strong>public String toLowerCase()</strong></td>
<td>Returns this String in lower case</td>
</tr>
<tr>
<td><strong>public String toUpperCase()</strong></td>
<td>Returns this String in upper case</td>
</tr>
</tbody>
</table>
Examples using String Methods

- **Really important** to remember: Strings are *immutable* (???)

- Getting length

  ```java
  String school = "University of Puget Sound";
  int size = school.length(); // size gets 25
  ```

- Find and replace (case-sensitive)

  ```java
  String name = "Adam A. Smith";
  String newName = name.replace("A. ", "");
  System.out.println(name); //still Adam A. Smith
  System.out.println(newName); //Adam Smith
  ```
Examples using String Methods (Cont.)

- Extracting a Substring

```java
String fullname = "Brad Richards";
String firstname = fullname.substring(0, 5);
```

- Extract Brad's last name:

```java
String fullname = "Brad Richards";
String lastname = fullname.substring(??? , ???);
```
Example

- Puget Sound email addresses are formed using first initial appended to the last name appended to @pugetsound.edu.

- Write an email address creation method called `getEmail`:
  - Two input parameters: first name, last name
  - Returns a Puget Sound email address in lowercase
    - Assume first and last names are never input as empty strings or null

- Example Usage:

```java
String myEmail = this.getEmail("David", "CHIU");
System.out.println(myEmail); // outputs dchiu@pugetsound.edu
```
Your Turn!

- Write a method called `vowelsAtEnds`:
  - Inputs a word (String)
  - Returns true if word starts and ends with vowel, false otherwise
    - Assume standard vowels only: a,e,i,o,u
    - Don't assume word will be given in lower case

  - Example Usage:

```java
System.out.println(this.vowelsAtEnds("Ada"));       // true
System.out.println(this.vowelsAtEnds("ice cream"));  // false
System.out.println(this.vowelsAtEnds("UMBRELLA"));  // true
System.out.println(this.vowelsAtEnds(""));          // false
System.out.println(this.vowelsAtEnds(null));         // false
```
Conclusion

- Abstraction is divide and conquer in software
  - Break up big problem into small, manageable pieces
  - Make sure you do a good job programming those pieces
  - Orchestrate together later to solve bigger problem
  - One of the important concepts in CS

- We also saw primitive types and their operators
  - What about object types? What are their operators? (Next)
Reminders:

- Hwk 3 (OrcaCard) due next Wednesday
- Exam next Thursday

New slides posted

Last time...

- Classes as data types; object variables are just pointers
- Object equality: reference equality vs. content equality

Today:

- Object composition: Abstraction and Modularity
Hwk 2 graded

- Really good work overall, esp. with if-then-else in divide()
- Weaknesses:
  - Unnecessary fields, code reuse in places, exchange() method
  - Inconsistent Javadocs commenting, e.g., no usage of @param or @return

Lab 5 post-mortem

- Takeaways: Focus was on class types and objects
  - "The dot notation" (control + space) to tell objects what to do
  - Reference equivalence and content equivalence
    - Dealing with null with != and ==
    - Dealing with .equals(...) <-- this needs some work
行政事务 2/21

- 上次课...
  - 抽象和模块化（那到底是什么？）
  - 编写复杂类不应令人感到害怕！
    - 分而治之！！！！
    - 组织体，大脑，胃
  - 阅读API

- 今天:
  - 最常用的对象：字符串
  - 字符串API，以及字符串操作