TA Hours Availability

• Since Saturday, the TAs have been seeing about 50-60 students daily at conceptual hours and a total of 178 sessions at debugging hours, of 196 sign-ups
• This year, we monitor the hours attendance and reallocate TAs between conceptual and debugging with the goal of seeing as many students as possible
  o Last year during Cartoon, each day there were about 50 students who did not get seen, so our goal is to make a major improvement
• If you aren’t seen multiple days in a row at debugging hours, please reach out to the HTAs

Collaboration Policy Reminder

• Brown’s policy is CS15’s policy: All written work must be your own
• We use MOSS to check for plagiarism
• MOSS is not fooled by renaming, resequencing, etc: it does a structural analysis
  o We carefully hand check high MOSS scores, not completely automated
• Regret Policy
• If ever in doubt, an NC and redo is far better than a directed NC on your transcript and parental notification

Lecture 11
Loops

“Life is just one damn thing after another.”
- Mark Twain

“Life isn’t just one damn thing after another… it’s the same damn thing over and over and over again.”
- Edna St. Vincent Millay
Outline

- **Turtle**
- **Looping**
  - **while Loops**
    - Lecture Questions: Slides 21 and 26
  - **for Loops**
  - **Choosing the Right Loops**
    - Lecture Questions: Slides 36 and 45

Introduction to Turtle (1/2)

- Before we see loops, we need some tools
  - We will use a Turtle to help us understand loops
  - Turtles are based on Seymour Papert’s Logo*, a language for beginners
  - Turtles are imaginary pens that when given instructions can draw shapes for us

Footnote:


Introduction to Turtle (2/2)

- Turtles know where they are and what direction they are facing and can move and turn
- Turtles can draw lines behind them as they move around the screen or just move without drawing
- PaneOrganizer holds instructions for the turtle
  - reminiscent of our first Robot example...
public class Turtle {
    // instance variables elided
    /* constructor for Turtle instantiates a Polygon
    representing the Turtle graphically */
    public Turtle() {
        // some code here
    }
    /* reset turtle to center of pane */
    public void home() {
        // some code here
    }
    /* turn right a specified number of degrees */
    public void right(double degrees) {
        // some code here
    }
    /* turn left a specified number of degrees */
    public void left(double degrees) {
        // some code here
    }
    // continued
    /* move forward a specified distance, drawing a line as the
    turtle moves */
    public void forward(int distance) {
        // some code here
    }
    /* move backward a specified distance, drawing a line as
    the turtle moves */
    public void back(int distance) {
        // some code here
    }
    /* move turtle to a specified position without
drawing a line */
    public void setLocation(Point2D loc) {
        // some code here
    }
    /* return turtle's location */
    public Point2D getLocation() {
        // some code here
    }
    /* returns the Polygon (the triangle) contained in Turtle
    class so that we can graphically add it in the P.O.*/
    public Node getNode() {
        // some code here
    }
}

Drawing with Turtle (1/2)

- Need class to tell Turtle how to draw some basic shapes
  - will contain a Pane and a Turtle
  - will have methods for each shape we want to draw
- First, determine what shapes we want
  - this lectures: square, random walk
Drawing with **Turtle** (2/2)

- How will we code it?
  - create `PaneOrganizer` class which defines methods for drawing each shape
  - `PaneOrganizer` also instantiates the root pane that the Turtle will draw on and contains the Turtle. The root is returned in `getRoot()`.
  - Turtle is a wrapper class that contains a polygon (a triangle) and defines methods for how the Turtle will move; it can also return its polygon as a node via `getNode()`.

```java
public class PaneOrganizer {

    // draw each shape
    private Turtle _turtle;
    private Pane _root;

    public PaneOrganizer() {
        _root = new Pane();
        _turtle = new Turtle();
        _root.getChildren().add(_turtle.getNode());
    }

    public Pane getRoot() {
        return _root;
    }
}
```

Note: Because this is a very small program, our logic is also in our `PaneOrganizer` class rather than a top-level logic class like we do in CS15 projects.

A Repetitive Solution (1/2)

- Let’s write `drawSquare` method in the `PaneOrganizer` class
- Brute force: write line of code for each side of the square

```java
public void drawSquare(int sideLen) {
    _turtle.forward(sideLen);
    _turtle.right(90);
    _turtle.forward(sideLen);
    _turtle.right(90);
    _turtle.forward(sideLen);
    _turtle.right(90);
    _turtle.forward(sideLen);
}
```

A Repetitive Solution (2/2)

- What if we wanted to make a more general method that handles regular shapes such as pentagons or octagons?
  - need to call `forward()` and `right()` for each side
  - cannot fix how many sides we need in generic method
  - note that we’re using the **Turtle**’s primitive methods to generate higher-level shapes that are normally already defined in JavaFX
- There must be an easier way!
Looping (1/2)

- Execute a section of code repeatedly
  - uses booleans (true and false) as loop conditions; continues looping as long as condition is true, but when boolean is false, loop condition equals exit condition and loop is terminated
  - as with conditionals, code in loop can be a single line or many lines enclosed in curly braces
  - section of code executed is called loop’s body

Looping (2/2)

- Three loop structures in Java
  - while loop
  - do while loop
  - for loop
- Differ in relation between body and loop condition, as well as length of execution
- Let’s look at while loop first

The while loop (1/2)

- Executes while stated condition is true
  - tests loop condition before executing body
  - if loop condition is false first time through, body is not executed at all

while (<loop condition>) {
  <loop body>
}

3/2/21
The **while** loop (2/2)

- Examples of loop conditions:
  - numClasses < 6
  - peopleStanding <= maxPeople
  - this.checkAmount() <= acctBalance
  - this.isSquare() //predicate, a method that returns a boolean
- Follows the same rules as conditions for **if-else** statements
- Multiple conditions can be combined using logical operators (and (&&), or (||), not (!))
  - (numClasses >= 3) && (numClasses <=5)
  - (peopleStanding <= maxPeople) || (maxPeople < 50)

---

**while** loop Flowchart (1/2)

- **while** loops continue while the loop condition is true
- **<loop condition>** can be any Boolean expression

---

**while** loop Flowchart (2/2)

- **while** loops continue while the loop condition is true
- **<loop condition>** can be any Boolean expression
  - Walk into the Ratty.
  - Is The Ratty open? 
    - Yes: Get more food.
    - No: Get more food.
  - Go to Jo's.
All Flow of Control Structures: 1-in, 1-out

- Benefits of predictable flow of control:
  - much easier debugging
  - compiler can optimize much better
- Different from "spaghetti" code (unorganized and difficult to maintain code) by having a goto which allows the program to jump to another line of code (based on h/w's unconditional and conditional "jump" instructions)
  - Go To Statement Considered Harmful letter by Edsger Dijkstra, CACM, 1968
  - IF-ELSE, etc., are "structured flow-of-control"

So, just how bad is goto?

Syntax: Random Walk Using while

- Method of PaneOrganizer class:
  - draws random lines while _turtle is within its pane
  ```java
  public void randomWalk() {
    // while _turtle's position is inside its pane, move _turtle randomly
    // _turtle's initial location set to (0,0)
    while (_root.contains(_turtle.getLocation())) {
      _turtle.forward((int) (Math.random() * 15)); // cast to [0-14]
      _turtle.right((int) (Math.random() * 360)); // cast to [0-359]
    }
  }
  ```
- On last step of walk, _turtle will move forward out of pane
  - the line is clipped by JavaFX since we don’t explicitly tell it to wrap around
  - no point in continuing to walk outside the pane
Lecture Question 1
What is the value of tempSum after this while loop is terminated?
```java
int tempSum = 0;
while (tempSum < 10) {
    tempSum += 3;
}
```
A. 10  
B. 9  
C. 12  
D. The loop will never terminate

The do while Loop
- do while always executes loop body at least once by switching order of test and body
- <loop condition> is Boolean expression

Example: Another Random Walk
- Method of PaneOrganizer class:
  o draws random lines while turtle is within pane
  o _turtle starts in center of root pane, so first step guaranteed to be within pane
```java
public void centeredRandomWalk() {
    // moves turtle to pane's center
    _turtle.home();
    // moves turtle randomly within pane
    do {
        _turtle.forward((int)(Math.random() * 15));
        _turtle.right((int)(Math.random() * 360));
    } while (_root.contains(_turtle.getLocation()));
}
```
Note the semicolon at the end of while statement
**do while vs. while (1/2)**

- In both loops:
  - stops executing body if loop condition is false
  - must make sure loop condition becomes false by some computations to avoid an “infinite loop”
  - infinite loop means your loop condition will never turn false — i.e., exit condition never occurs (and your program “freezes up”!)  

**do while vs. while (2/2)**

- **do while**
  - body always executes at least once
  - loop condition tested at bottom of loop body
- **while**
  - body may not execute at all
  - loop condition tested before body; loop condition variables must be set before loop entry
  - useful for screening bad data that might cause statements within loop to fail (e.g. while (ref != null))

**Lecture Question 2**

What’s the difference between these two loops?

Loop 1:
```java
while(andyIsAway()) {
    _tas.takeADayOff();
    _tas.takeADayOff();
}
```

Loop 2:
```java
do {
    _tas.takeADayOff();
    _tas.takeADayOff();
} while (andyIsAway());
```

A. In the second loop, the condition is tested before the body
B. In the second loop, the TAs always take at least 1 day off
C. In the first loop, the body is executed before the condition is tested.
D. There is no difference between the two loops
for loops (1/4)

- Most specialized loop construct (and the first high-level, goto-less loop in FORTRAN): typically used to execute loop body a predetermined number of times
  - while and do while loops can execute body for undetermined number of times; based on boolean
- This is the syntax for a for loop:
  ```
  for (<init-expr>; <loop condition>; <update>) {
      <loop body>
  }
  ```

for loops (2/4)

```
for (<init-expr>; <loop condition>; <update>) {
    <loop body>
}
```

- <init-expr>
  - expression for setting initial value of loop counter (traditionally use single char. identifier; e.g., i)
  - executed at start of loop code, only once, not for each time through the loop

for loops (3/4)

```for (<init-expr>; <loop condition>; <update>) {
    <loop body>
}
```

- <loop condition>
  - true or false
  - test involves loop counter to determine if loop should execute
  - checked at start of every loop (including the first)
for loops (4/4)

for (<init>-expr>; <loop condition>; <update>) {
    <loop body>
}

- <update>
  - expression that modifies loop counter
  - run at end of every <loop body>, just before returning to the top of the loop

drawSquare Revisited

- Better way of drawing square than explicitly drawing each side:

```java
public void drawSquare(int sideLen) {
    /* start with integer i initialized to 0;
       execute as long as i < 4; each execution increments i by 1 */
    for (int i = 0; i < 4; i++) {
        _turtle.forward(sideLen);
        _turtle.right(90);
    }
}
```

for Flowchart

- for loop has four parts
  - initialize value of counter
  - test loop condition
  - loop body
  - update counter

Is <loop condition> True?

Yes ---> <loop body>

No ---> <update-counter>

<update-counter> ---> <init-counter>

<init-counter> ---> previous statement

<rest of program>
for Flowchart

- We can use an example of a student reading books on different floors of the SciLi.

```java
Student student = new Student("Huey");
student.goToSciLi();
for (let floor = 1; floor < 14; floor++) { 
    student.readBook(); //read a new book
}
student.goHome();
```

Note: For this example, we use the old SciLi, where every floor had books!

34

Choosing the Right Loop (1/2)

- `for` loop is called a **definite** loop because you can typically predict how many times it will loop.
- `while` and `do while` loops are **indefinite** loops, as you do not know when they will end.
- `for` loop is typically used for math-related loops like counting finite sums.

35

Choosing the Right Loop (2/2)

- `while` loop is good for situations where boolean condition could turn `false` at any time.
- `do while` loop is used in same type of situation as `while` loop, but when code should execute at least once.
- When more than one type of loop will solve problem, use the cleanest, simplest one.

36
Lecture Question 3
What is the value of \( \text{\texttt{sum}} \) at the end of the following loop?

\[
\text{\texttt{sum}} = 0; \\
\text{\texttt{for}} (\text{\texttt{i}} = 0; \text{\texttt{i}} <= 10; \text{\texttt{i}}+=2) \{} \\
\quad \text{\texttt{sum}}++; \\
\text{\texttt{}} \\
\]

A. 10  
B. 11  
C. 5  
D. 6

Syntax: Nested Loops
- Loops, as with \texttt{if} statements, can be \texttt{nested}!
- Example: \texttt{drawFilledSquare}

\begin{verbatim}
public void drawFilledSquare(int \texttt{sideLen}) { 
    // fill in concentric squares:
    for (int \texttt{i} = 0; \texttt{i} < (\texttt{sideLen}/2); \texttt{i}++) { 
        for (int \texttt{j} = 0; \texttt{j} < 4; \texttt{j}++) { 
            _turtle.forward(\texttt{sideLen} – (2*\texttt{i})); 
            _turtle.right(90); 
        } /* note we can use loop counter \texttt{R/O (read-only)} 
        in loop but never reset it there! */ 
        // position turtle for next iteration:
        _turtle.right(90); 
        _turtle.forward(1); 
        _turtle.left(90); 
        _turtle.forward(1); 
    } 
}
\end{verbatim}

- What does this do?
  - decrementing \texttt{sideLen} by 2 each iteration to guarantee 
    that each “inner square” 
    drawn in the inner loop is 
    exactly one unit away on 
    either side from square 
    immediately “outside” of it 
    (hence, one + one = two)

Syntax for Nested Loops Explained
- Turtle is represented by ▲
- What is the outer loop doing?
  - first draws outer square
  - Turtle starts upright!
  - Rotate 90 degrees right!
  - Move forward 1 unit!
  - Rotate 90 degrees left!
  - Move forward 1 unit!
  - Draw inner square

\texttt{drawFilledSquare} draws concentric squares; each individual square is drawn using the nested loop.
Looping to Make a Filled-in Square

- 3D Printing a Pizza
  
  https://www.youtube.com/watch?v=ISXgC-YPnpc

Decrementing Counter

- We can count backwards in our loop too
  - in fact, we can update however we want
    
    ```java
    public void countDownSeconds()
    {
        /*change counter to decrement, and change the loop condition accordingly */
        for(int i = 5; i > 0; i--){
            System.out.print(i);
        }
    }
    
    Output: 54321
    ```

- **for** loops end in one of two ways
  - when counter value equals limit (for < or >)
  - when counter value “goes past” limit (for <= or >=)
  - thus, countDownSeconds() would also print 0 if used i => 0
  - beware of such “off-by-one” errors! → hand simulation really helps!

  ```java
  // Execution continues here after loop is done or after break statement is executed
  ```

- **break**
  - break causes immediate exit from a flow-of-control structure (e.g., switch, while, do while, for)
  - Example:
    ```java
    for (int i = 0; i < 10; i++) {
        if (_cookieJar.getNumberOfCookies() == 0) {
            break; //If there are no cookies left, we should break out of the loop!
        }
    }
    this.eatACookie();
    ```

  - Execution continues with first line of code after structure
  - There are other ways to do this loop...

  ```java
  // Execution continues here after loop is done or after break statement is executed
  ```
continue

- When used in `while`, `for`, or `do while` structures, `continue` skips remaining statements in body of that structure and proceeds with next iteration of loop.
  - Useful if there is list of data that you are looping over and you want to skip processing of data that is somehow “not legal”.
- In `while` and `do while` structures, execution continues by evaluating loop-continuation condition.
- In `for` structure, execution continues by incrementing counter and then evaluating loop condition.

Example

// We’d like to try on swimsuits that hang on a rack
for (int i = 0; i < 20; i++) {
  if (!rack.isSwimsuitOnHanger(i)) {
    // If there’s no swimsuit on the current hanger,
    // skip to the next iteration
    continue;
  }
  // Only do this if there’s a swimsuit on the hanger
  this.tryOnSwimsuit(rack.getSwimsuitOnHanger(i)); // Get swimsuit and try it on
}

// more code here

Boolean Predicates and Flags

- A **Boolean predicate** is a method that returns a `boolean` (e.g., `isDone`, `isAvailable`, `isSwimsuitOnHanger(i)`)
- A **Boolean flag** records the result of a predicate: set and saved in one place, used later in different place.
- Example (implementing a `for` loop, using `while`):

```java
boolean isDone = false;
int i = 0;
while (!isDone) {
  i++;
  if (i == 5) { isDone = true; }
}
```

Note: Here, the Boolean flag is set within loop, which, though legal, is not practical.
Lecture Question 4

In the loop to the right, what is the value of i upon exit?

A. 4  B. 5  C. 6  D. Infinite loop

```java
boolean isDone = false;
int i = 0;
while (!isDone){
i++;
if(i == 5){
isDone = true;
}
}
```

Empty Intervals

- Example scenario: we want to keep a running sum of a sequence of numbers
- What happens if we try to add integers in this loop?

```java
public int sum() {
    int tempSum = 0;
    for (int i = 1; i < 1; i++) {
        tempSum += i;
    }
    return tempSum;
}
```

- Answer: body of loop is not executed
- Why?
  - loop condition is false for initial counter value

Correct Example

- What about this loop?

```java
/*This method sums all numbers from 1 up to and including 10 */
public int sum() {
    int tempSum = 0;
    for (int i = 1; i <= 10; i++) {
        tempSum += i;
    }
    return tempSum;
}
```

- It will work!
Off-by-one Errors

- These errors occur when loop executes one too many or one too few times
  - example: add even integers from 2 to some `number`, inclusive
    ```java
    count = 2;
    result = 0;
    while (count < number) {
        result += count;
        count += 2;
    }
    ```
  - Should be:
    ```java
    while (count <= number) {
        ...
    }
    ```
  - Produces incorrect result if `number` is assigned an even value. Values from 2 to `number-2` will be added (i.e., `number` is excluded)
  - Now, value of `number` is included in summation

Syntax: Other Loop Errors (1/2)

- Make sure test variables have proper values before loop is entered
  ```java
  ...
  int product = 0;
  do {
      product *= 2;
  } while (product < 100);
  ```
- Make sure tests check proper conditions
  ```java
  ...
  for (int i = 1; i <= 100; i += 2) {
      // do something here
  }
  ```
  - Will we ever get here?

Lecture Question 5

Given the following code:
```java
int num = 2021;
do {
    num--;
} while (num < 2021);
```
What do you expect will happen?
A. Loop will never end
B. Loop will run 2021 times (until `num` is 0), then end
C. Loop will run only once
Syntax: Other Loop Errors (2/2)

- ALWAYS HAND SIMULATE first, last, and typical cases through a loop to avoid off-by-one or infinite loop errors
  - the first and last cases of a loop’s execution are called boundary conditions or edge cases or corner cases
  - hand simulation doesn’t just apply to loops – use it for everything! Trust us – it saves debugging time!

Which loop to use?

- You want to stack 17 sandwiches
- Your job is to stand at the end of the bowling alley and pick up all the pins, one by one, that have been knocked over
- Sleep until your clock reads 7:51AM or later

Announcements

- Cartoon On-Time deadline tomorrow 3/3 @ 11:59 PM
- No section this week!
  - Instead, you must watch the Debugging Video and take the short Debugging Quiz by Saturday EOD
- DoodleJump out Thursday
  - Start early! There is a code checkpoint due at next week’s section
MEDIA, DATA SCIENCE, AND SOCIAL CHANGE: A CONVERSATION WITH HARMONY LABS

March 5 @ 12:00 pm - 1:00 pm

Uses of ML & DL

- Works based on pattern recognition → applications everywhere
  - Voice Recognition (e.g., Siri, Alexa)
  - Image Recognition/Classification (e.g., Microsoft Seeing AI)
  - Facial Recognition
  - Text generation/writing* (e.g., GPT-3)
  - Weather Forecasting
  - Translation/Internationalization (e.g., Google Translate)
  - Content recommendations/"Personalization" (e.g., Netflix, YouTube, Google Search)
  - Autocomplete
  - Content moderation (recall FB)
  - Self-Driving/Autonomous Vehicles (e.g., Tesla)
  - and more, and more, and more!

"[ML] uses past data and projects it forward" – danah boyd 00, in visiting talk for CS1951V Hypertext Seminar

Positive Use Case: Improving Accessibility!

- Sound-based User Interfaces (UIs)
  - Blind/low-vision people often cannot use standard Graphical User Interfaces (GUIs), instead use Voice Recognition/Generation
  - Examples: Noavia, Oregon
- Auto-captions for video
  - Deaf/Hard of Hearing people (and more!) rely on closed captions to understand video
  - Examples: Google Live Caption, Otter.ai (for Zoom and Google Meet!)
- Live image recognition (in development)
  - In day-to-day life, many blind/low-vision people cannot access print documents, visual cues in surroundings
  - Example of "Scene Recognition" functionality, Microsoft Seeing AI

Problematic Application: Surveillance (1/2)

- Facial recognition now widely used for authentication (opt-in process, e.g., Apple FaceID)
  - interesting ramification: makes it easier for law enforcement to unlock suspects’ phones without express consent
- Law enforcement has used Facial Recognition (FR) for decades with limited data (government databases: ~411M photos)
  - limited, biased data → false matches → false allegations/arrests
- Private companies allowed to use more data → make & sell FR for US law enforcement (legal loophole)
  - still, limited & unequal accuracy – e.g., 2018 scandal: Amazon AI system fail to identify members of Congressional Black Caucus

Source: Getty Images, via Vox
Problematic Application: Surveillance (2/2)

- **2019:** new company, Clearview AI, uses images from social media, public websites for facial recognition (~3B photos)
  - 100% legal under US regulations, but against social media fair use/scraping policies
  - even if taken down online, images not removed from database
- **2021:** Clearview AI illegal in Canada due to severe privacy infringement
  - but Clearview will not delete Canadians' data
- Clearview AI still used by 2,400 US law enforcement agencies

Source: Clearview AI, via NY Times (Jan. 18, 2020)

Different Applications, Overlapping Issues

“Technology is neither good nor bad, nor is it neutral.”
- Martin Kranzberg, technology historian

- ML-based
- rely on large datasets
- ongoing data collection
- use image recognition
- pose privacy concerns
- used by law enforcement
- based on non-consensual data collection
- improper use jeopardizes privacy, safety, bias

- used by private individuals
- consent from primary users
- improper use causes little harm