# CSE 12: Week 1 Discussion

Focus: PA 1, Logistics, JUnit, interface

# About me

Qiyue (Cheery) Wang

BS in Computer Science from UCSD

First year Master student

Email: qiw131@ucsd.edu

Tutor hour: Wed 1 - 2 pm PST (subject to change)

# What will Discussion cover?

- Introduction to the Programming Assignment due next week (every **Wednesday at 11:59 pm PST**)
  - What you are asked to do
  - Related CS concepts
- Answers to course related questions (if you have any)

# How to look for PA help?

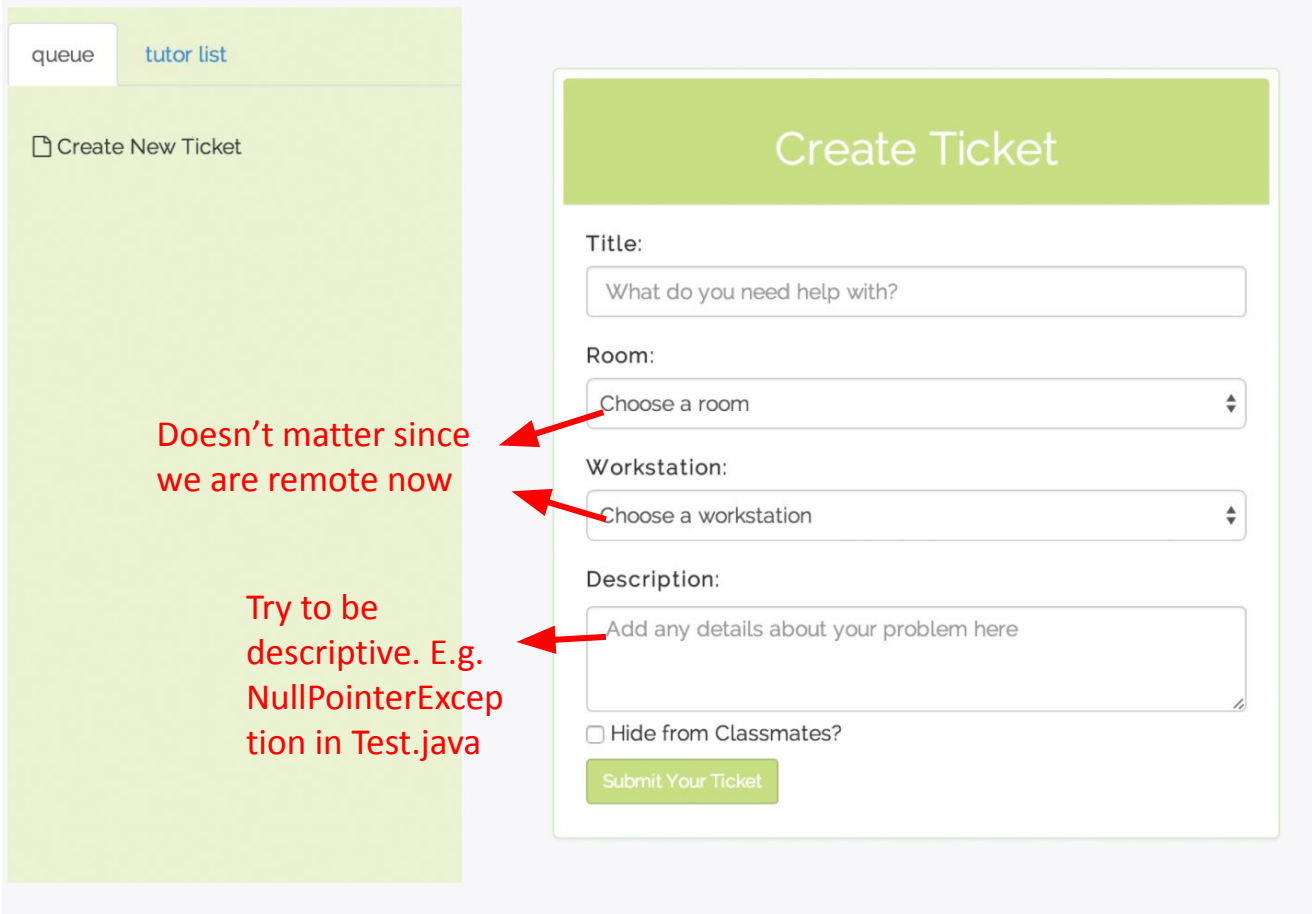
- Course slides/code, discussion slides/recording
  - Slides can be found in schedule from course website
- Optional ZyBook
- Tutor hours
- Professor office hours (more focused on general concepts than debugging)

# Poll: Have you gone to tutor hours before?

A. I have gone to CSE 12 tutor hours
B. I have only gone to tutor hours for other CSE courses at UCSD
C. I have never gone to tutor hours for any CSE course

# How to use tutor hours?

- Instructions on course website
- Go to Autograder
- Login with UCSD SSO credentials (@ucsd.edu email and MyTritonlink password)
- Create ticket and wait for help


queue
tutor list
Create New Ticket
Create Ticket
Title:
What do you need help with?
Room:
Choose a room
Workstation:
Choose a workstation
Description:
Add any details about your problem here
Hide from Classmates?
Submit Your Ticket
Doesn't matter since we are remote now
Try to be descriptive. E.g. NullPointerException in Test.java

# What's expected from YOU in tutoring

- Enter tutor's zoom room as soon as your ticket is accepted and the link is posted
  - If you do not enter the zoom room in two minutes after the link is posted, your ticket will be skipped, and the next available tutor will accept your ticket
  - If you are already skipped once, and you do not join the zoom room in time again, your ticket will be cancelled
- Only ask one question for each ticket
  - If you have more than one question, you can always create another ticket right after your previous question is resolved
- Try debugging by yourself first (e.g. print statements, debugger, etc.) before asking for help
- 5-minute rule: Tutors only help you unstuck. They are not expected to solve the problem for you!

# PA1: Testing Shopping Baskets

# Overview

https://github.com/CSE12-SP21-Assignments/cse12-sp21-pa1-Testing

You are working for a Web shopping company where you need to create shopping cart functionality to keep track of items before a customer checks out.

- You are given 13 different implementations of **Basket**
- Your job is to write JUnit tests to cover the potential issues that might occur in the implementations

Note: All of your code will be written in **BasketTest.java**. DO NOT change any other file

# Overview

- Part 1
  - Write your tests for the **Basket** implementations
  - Submit **BasketTest.java** to “Programming Assignment 1 - code” on Gradescope
- Part 2
  - Answer the questions in the write up on the Gradescope assignment “Programming Assignment 1 - questions”

# Grading

For this assignment, your code's grade is dependent on the number of **Basket** implementations you successfully distinguished by unit tests.

Example 1: 3 **Basket** implementations distinguished

| | addOneItem() | addDiffItems() |
|---|---|---|
| Basket1 | Success | Success |
| Basket2 | Success | Failure |
| Basket3 | Failure | Success |

# Grading

Example 2: 2 **Basket** implementation distinguished

| | addOneItem() | addDiffItems() |
|---|---|---|
| Basket1 | Failure | Success |
| Basket2 | Success | Success |
| Basket3 | Failure | Success |

Note: In this assignment, an exception (red cross on Eclipse) is ok for distinguishing implementations. **However**, in general, we don't want exceptions to count as failure, only for this PA

# Grading

- For this assignment, style will not be graded
  - For future assignments, style WILL BE graded
- Style requirements for future assignments
  - Try to keep each line of code fewer than 100 characters
  - Consistent indentation
  - Meaningful/Descriptive names for methods and tests
    - Examples of not meaningful names: test1, method1

# Practice: How to write unit tests

Given a class **`Student`** with the method **`addCourse`** that adds a **`Course`** to a student's course list, **`courses`**.

**What are some unit tests we can create to make sure any given implementation is correct?**

## Class Course

| | |
|---|---|
| **Course**(**String** name)<br>**Constructor** that creates a course specified by its name. | |
| String | **getName**()<br>Returns name of the Course object. |
| boolean | **isEqualTo**(Course other)<br>Returns whether or not a Course object's name is the same as another. |

## Interface Student

| | |
|---|---|
| String | **getName**()<br>Returns name of an object of a class that implements the Student interface. |
| Course[] | **getCourses**()<br>Returns course list of an object of a class that implements the Student interface. |
| void | **addCourse**(Course course)<br>Adds a course to a course list of an object of a class that implements the Student interface. |

## Class StudentX (implements Student)

| |
|---|
| **StudentX**(**String** name, **Course[]** courses)<br>**Constructor** that creates a student specified by its name and course list. |

# What are some test ideas for addCourse?

- Make sure list contains the added course after
- Make sure list length increases by 1
- Test capacity bounds
- Doesn’t affect previous information
- Check for duplicates
- Make sure added course is actually a course

# 2 implementations of `addCourse`

```java
public void addCourse(Course course){
      int size = courses.length + 1;
      Course[] newCourseList = new Course[size];

      for(int i = 0; i < courses.length; i++){
            if(course.isEqualTo(courses[i])){
                  return;
            }
      }

      for(int i = 0; i < size - 1; i++){
            newCourseList[i] = courses[i];
      }
      courses = newCourseList;

      return;
}
```

Missing adding **`course`** to **`newCourseList`**

```java
public void addCourse(Course course){
      int size = courses.length + 1;
      Course[] newCourseList = new Course[size];

      for(int i = 0; i < courses.length; i++){
            if(course.equals(courses[i])){
                  return;
            }
      }

      int i;
      for(i = 0; i < size - 1; i++){
            newCourseList[i] = course;
      }
      newCourseList[i] = course;
      courses = newCourseList;

      return;
}
```

Are there any more tests we should add?

# Source code - JUnit testing example

Given 3 versions of the **`Student`** interface in classes **`StudentA`**, **`StudentB`**, and **`StudentC`**, we can implement unit tests to check for functionality. These will be added to the **`StudentTest`** class (similar to **`BasketTest`** class in PA1).

Note: All code will be posted as well as a skeleton version of **`StudentTest`** so that you can try writing your own

# Additional Resource

Review worksheet for Java interface:

- Interface worksheet
- Interface worksheet solution

Questions?

# Debugging

# Types of Errors

- Compile Errors
- Runtime Errors
- Logic Errors

# Compiler Errors

- ***Syntax Error***
  - Error in usage of Java
  - Detected by the compiler
  - A program with compilation errors cannot be run
- ***Syntax Warning***
  - Warning message generated by the compiler
  - The program can be run

# Compiler Errors

- Very common (but sometimes hard to understand). Examples of syntax errors:
  - Forgetting a semicolon
  - Leaving out a closing bracket }
  - Re-declaring a variable
  - Others?
- Hint to help find/fix compiler errors
  - Compiler errors are cumulative: when you fix one, others may go away
  - Read the error messages issued by the compiler!

```java
public class Game {

  Public Static Void main(String args[]) {

    System.out.println("If I choose Paper,");
    System.out.println("And you choose Scissors,");
    System.out.println("Then I win, and you lose!")

  }
```

```
$javac Game.java
Game.java:3: error: ';' expected
  Public Static Void main(String args[]) {
                ^
Game.java:7: error: ';' expected
    System.out.println("Then I win, and you lose!")
                                                   ^
Game.java:9: error: reached end of file while parsing
  }
   ^
3 errors
```

# Runtime Errors

- ***Runtime Error***: program runs but gets an *exception* error message
- Program may be terminated
- Very common runtime errors are:
  - Null reference (NullPointerException)
    - No object is referenced by the reference variable, i.e. it has the value null
  - Array index out of bounds (ArrayIndexOutOfBoundsException)
  - Running out of memory
    - From creating a new object every time through an infinite loop

```java
public class Example {

    public static void main(String[] args) {
        Object obj = null;
        obj.hashCode();
    }

}
```

```
Exception in thread "main" java.lang.NullPointerException
        at Example.main(Example.java:5)
```

```java
public class Example {

    public static void main(String[] args) {
        int[] array = new int[5];
        // ... populate the array here ...
        for (int index = 1; index <= array.length; index++)
        {
            System.out.println(array[index]);
        }
    }

}
```

```
0
0
0
0
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at Example.main(Example.java:8)
```

# Logic Errors

- Programs run but results are not correct
- Caused by incorrect algorithm
- Very common logic errors are:
  - Using == instead of equals method
  - Infinite loop
  - Misunderstanding of operator precedence
  - Starting or ending at the wrong index of an array
  - Misplaced parenthesis
    - Keep in mind the scope of the variables! (instance variables, formal parameters, local variables)

# Debugging Strategies

- Trace your code by hand
- Put print statements to inspect variable values or use the debugger in your IDE

Questions?