# CSE 11
# Accelerated Intro to Programming
# Discussion Section 4

Ayon Biswas, Spring 2021

This discussion is being recorded

# Logistics

- PA3 is due today
- PA4 released today
- PA3 resubmission will launched tomorrow and due after 2 weeks

# Interface

- Classes that share behaviour
- An interface is declared by using the interface keyword
- all the methods in an interface are declared with the empty body, For Example-

```
interface /* Interface Name */ {
    /* Method Signature */;
    /* Method Signature */;
    /* ... */
}
```

- classes and abstract classes can implement interfaces with the following syntax -

```
class /* Class Name */ implements /* Interface Name */ {
    /* ... */
}
```

- interface allowed us to treat multiple classes as a shared type, For Example - We use it to create Unions of regions without worrying about what the underlying Region type actually was

# Example

```
interface Region {
     boolean contains(Point p);
}
class SquareRegion implements Region {
 ...
     public boolean contains(Point toCheck) {...}
}
class CircleRegion implements Region {
 ...
     public boolean contains(Point toCheck) {... }

class UnionRegion {
 Region r1, r2;
 UnionRegion(Region r1, Region r2) {
     this.r1 = r1;
     this.r2 = r2;
 }
 public boolean contains(Point p) {
     return this.r1.contains(p) || this.r2.contains(p);
 }
}
```

# Abstract class

- Classes that share implementation - even fields and method bodies
- Abstract classes are used whenever there is code duplication
- declared using “**abstract**” keyword

```
abstract class /* Class Name */ {
    /* Shared code */
  }
```

- An abstract class has no use until unless it is extended by some other class
- classes can extend abstract classes with the following syntax:

```
class /* Class Name */ extends /* Abstract Class Name */ {
    /* ... */
}
```

# Example :

```
abstract class ComboRegion implements Region {
    Region r1;
    Region r2;
    ComboRegion(Region r1, Region r2) {
        this.r1 = r1;
        this.r2 = r2;
    }
 }
class UnionRegion extends ComboRegion {
    UnionRegion(Region r1, Region r2) {
        super(r1, r2);
 }
   public boolean contains(Point p) {
        return this.r1.contains(p) || this.r2.contains(p);
    }
 }
class IntersectionRegion extends ComboRegion {
    IntersectionRegion(Region r1, Region r2) {
        super(r1, r2);
 }
   public boolean contains(Point p) {
        return this.r1.contains(p) && this.r2.contains(p);
    }
 }
```

# Tester

- import tester.*;
  - tester.jar – java archive
    - Libraries that contain classes that we can use in our own code
      - Tester
- Tester class allows us to create methods to unit test our code
  - Unit testing – compare actual values versus expected values
    - t.checkExpect(<actual value>, <expected value>);
    - test method name should begin with “test”
  - Goal: get all tests to pass
    - Confidence that your code/solution is correct

# PA4

- **Collaboration Policy PA4**

Different assignments in this course have different collaboration policies. On this assignment, **you cannot share code publicly on Piazza or with any other students in the course.** If you need to share code, ask in a private Piazza post or in 1-on-1 hours. Still *do* ask any public code questions about lecture code, quizzes that are past, or other conceptual questions, just no code from this PA.

- https://github.com/CSE11-SP21-Assignments/cse11-sp21-pa4-starter
- Due next Wednesday at 11:59pm PST
- Lots of stuff going on. Seek help if necessary
- Start early!

# Thanks!