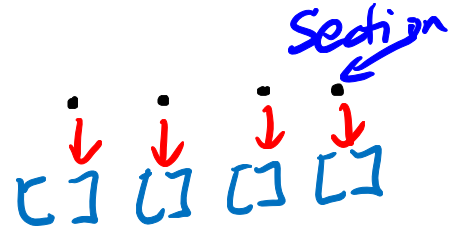


## 1 ADT Selection

Suppose we have a TA Shreyas who teaches multiple discussion sections! A student may frequent more than one discussion section. For each situation below, choose the best ADT(s) out of the following — Map, Set, List — and explain how you can use the ADT(s) to solve the problem. Each subpart is independent of the previous. One answer may involve multiple ADTs. There may be multiple efficient answers for each problem.



1. Storing all the Students in Shreyas's first section in alphabetical order.

**List - multiple students with the same first name**

2. Storing all the Students by their section, where Students within a section are sorted alphabetically.

**Map - key as the section, value as the students within that section (List)**

3. Storing the Students in *all* of Shreyas's sections. There shouldn't be duplicates.

**Set - List won't account for duplicates**

4. Quickly getting a Student by sid.

**Map - key as sid, value as Student**

5. Quickly getting all Students of a given name. Names aren't necessarily unique.

**Map - key as name, value as Students of that name (List)**

6. Cycling through the Students in one discussion section.

**List**

## 2 The ABCs of OOP

Indicate what each line the main program in class **D** would print, if the line prints anything. If any lines error out, identify the errors as compile time or runtime errors.

```

Animal a = new Dog();
Dog d = new Animal();

public class A {
    public void x() {
        System.out.println("Ax");
    }

    public void y(A z) {
        System.out.println("Ay");
    }
}

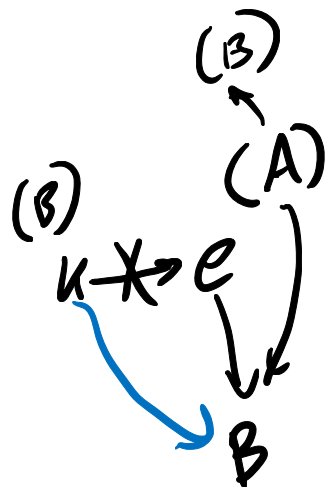
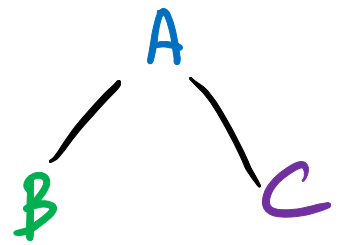
public class B extends A {
    public void y() {
        System.out.println("By");
    }

    public void y(B z) {
        System.out.println("Byz");
    }
}

public class C extends A {
    public void x() {
        System.out.println("Cx");
    }
    private int expected = 0;
}

public class D {
    public static void main(String[] args) {
        A e = new B();
        A f = new C();
        B g = new A();
        B h = new C();
        C i = (C) new A();
        B j = (A) new C();
        B k = (B) e;
        f.x();
        e.x();
        e.y();
        ((B) e).y();
        e.y(e);
        e.y(f);
    }
}

```



i.expected ?

E  
C  
R  
C  
**Cx**  
**Ax**  
C  
**By**  
**Ay**  
**Ay**

### 3 Classy Cats

Look at the `Animal` class defined below. The `protected` access modifier may be new to you. Simply put, it gives classes in the same package and subclasses access to those variables. Don't worry too much about understanding this - it's not in scope for exams.

```

1 public class Animal {
2     protected String name, noise;
3     protected int age;
4
5     public Animal(String name, int age) {
6         this.name = name;
7         this.age = age;
8         this.noise = "Huh?";
9     }
10
11    public String makeNoise() {
12        if (age < 2) {
13            return noise.toUpperCase();
14        }
15        return noise;
16    }
17
18    public String greet() {
19        return name + ": " + makeNoise();
20    }
21 }

```

- (a) Given the `Animal` class, fill in the definition of the `Cat` class so that it makes a "Meow!" noise. Assume this noise is all caps for kittens, i.e. Cats that are less than 2 years old.

```

1 public class Cat extends Animal {
2     public Cat(String name, int age) {
3         super(name, age);
4         this.noise = "Meow!";
5     }
6 }

```

- (b) "Animal" is an extremely broad classification, so it doesn't really make sense to have it be a class. Look at the new definition of the `Animal` class below.

```

1 public abstract class Animal {
2     protected String name;
3     protected String noise = "Huh?";

```

```

4     protected int age;
5
6     public String makeNoise() {
7         if (age < 2) {
8             return noise.toUpperCase();
9         }
10        return noise;
11    }
12
13    public String greet() {
14        return name + ": " + makeNoise();
15    }
16
17    public abstract void shout();
18    private abstract void count(int x);
19 }

```

Fill out the Cat class again below to allow it to be compatible with Animal (which is now an abstract class) and its two new methods.

```

1 public class Cat extends Animal {
2     public Cat() {
3         this.name = "Kitty";
4         this.age = 1;
5         this.noise = "Meow!";
6     }
7
8     public Cat(String name, int age) {
9         this();
10        this.name = name;
11        this.age = age;
12    }
13
14    @Override
15    public _____ void _____ shout() {
16        System.out.println(noise.toUpperCase());
17    }
18
19    @Override
20    private _____ void _____ count(int x) {
21        for (int i = 0; i < x; i++) {
22            System.out.println(makeNoise());
23        }
24    }
25 }

```