

# Discussion 5: Test Review

**Discussion:** Wed 4-5pm 120 Wheeler Hall

**Lab:** Thurs 3-5pm 275 Soda Hall

**OH:** Thurs 1-2pm in 220 Jacobs Hall

**email:** [asetoodehnia@berkeley.edu](mailto:asetoodehnia@berkeley.edu)

**website:** [asetoodehnia.github.io](http://asetoodehnia.github.io) (or find it through the CS 61B staff webpage)

*Please fill out this google form:* <http://bit.ly/61b-disc-5>

## Reminders:

- MIDTERM tomorrow 😬😄
- proj1 is out! Try and get started on this early (but don't worry about it until after your test)
  - definitely check out the [video walkthrough](#) if you haven't already

## Today's Goals:

1. Review for the midterm
2. Get through as many questions as possible
3. Questions and general advice for test-taking at the end

## 1 Inheritance Practice

```

public class Q {
    public void a() {
        System.out.println("Q.a");
    }
    public void b() {
        a();
    }
    public void c() {
        e();
    }
    public void d() {
        e();
    }
    public static void e() {
        System.out.println("Q.e");
    }
}

public class R extends Q {
    public void a() {
        System.out.println("R.a");
    }
    public void d() {
        e();
    }
    public static void e() {
        System.out.println("R.e");
    }
}

public class S {
    public static void main(String[] args) {
        R aR = new R();
        run(aR);
    }
    public static void run(Q x) {
        x.a();           /* Output: R.a */
        x.b();           /* Output: R.a */
        x.c();           /* Output: Q.e */
        ((R)x).c();      /* Output: Q.e */
        x.d();           /* Output: R.e */
        ((R)x).d();      /* Output: R.e */
    }
}

```

dynamic type is R  
static type is Q

In run, write what gets printed next to each line when it is called from main.

## 2 Reduce

We'd like to write a method `reduce`, which uses a `BinaryFunction` interface to accumulate the values of a `List` of integers into a single value. `BinaryFunction` can operate (through the `apply` method) on two integer arguments and return a single integer. Note that `reduce` can now work with a range of binary functions (addition and multiplication, for example). Write two classes `Adder` and `Multiplier` that implement `BinaryFunction`. Then, fill in `reduce` and `main`, and define types for `add` and `mult` in the space provided.

```
import java.util.ArrayList;
import java.util.List;
public class ListUtils {
    /** If the list is empty, return 0.
     *   If it has one element, return that element.
     *   Otherwise, apply a function of two arguments cumulatively to the
     *   elements of list and return a single accumulated value.
     */
    public static int reduce(BinaryFunction func, List<Integer> list) {
        if (list.size() == 0) { return 0; }
        int total = list.get(0);
        for (Integer i : list) {
            total = func.apply(total, i);
        }
        return total;
    }

    public static void main(String[] args) {
        ArrayList<Integer> integers = new ArrayList<>();
        integers.add(2); integers.add(3); integers.add(4);
        Adder add = new Adder();
        Multiplier mult = new Multiplier();
        reduce(add, integers); // Should evaluate to 9
        reduce(mult, integers); // Should evaluate to 24
    }
}

interface BinaryFunction {
    int apply(int x, int y);
}

//Add additional classes below:
public class Adder implements BinaryFunction{
    int apply(int x, int y){
        return x+y;
    }
}
public class Multiplier implements BinaryFunction{
    int apply(int x, int y){
        return x*y;
    }
}
```

### 3 Comparator

We'd like to sort an `ArrayList` of animals into ascending order, by age. We can accomplish this using `Collections.sort(List<T> list, Comparator<? super T> c)`. Because instances of the `Animal` class (reproduced below) have no natural ordering, `sort` requires that we write an implementation of the `Comparator` interface that can provide an ordering for us.

Note that an implementation of `Comparator` only needs to support pairwise comparison (see the `compare` method). Remember that we would like to sort in ascending order of age, so an `Animal` that is 3 years old should be considered "less than" one that is 5 years old.

```
1 public interface Comparator<T> {
2     /** Compares its two arguments for order.
3      * Returns a negative integer, zero, or a positive integer if the first
4      * argument is less than, equal to, or greater than the second. */
5     int compare(T o1, T o2);
6
7     /** Indicates whether some other object is "equal to" this
8      * comparator. (You don't need to implement this for this problem) */
9     boolean equals(Object obj);
10 }

1 import java.util.ArrayList;
2 import java.util.Collections;
3 public class Animal {
4     private String name;
5     private int age;
6     public Animal(String name, int age) {
7         this.name = name;
8         this.age = age;
9     }
10    /** Returns this animal's age. */
11    public int getAge() {
12        return this.age;
13    }
14    public static void main(String[] args) {
15        ArrayList<Animal> animals = new ArrayList<>();
16        animals.add(new Animal("Garfield", 4));
17        animals.add(new Animal("Biscuit", 2));
18        AnimalComparator c = new AnimalComparator(); //Initialize comparator
19        Collections.sort(animals, c);
20    }
21 }

import java.util.Comparator;
public class AnimalComparator implements Comparator< Animal > {
    public int compare(Animal o1, Animal o2) {
        return o1.getAge() - o2.getAge();
    }
}
```

## 4 Midterm Practice

---

```
public class PasswordChecker {
    /**
     * Returns true if the password is correct for the user
     */
    public boolean authenticate(String a, String b) {
        // Does something secret
    }
}

public class User {
    private String username;
    private String password;

    public void login(PasswordChecker p) {
        p.authenticate(username, password);
    }
}
```

Write a class containing a method `public String extractPassword(User u)` which returns the password of a given user `u`. You may not alter the provided classes. Note the access modifiers of instance variables.

```
public class PasswordExtractor extends PasswordChecker {
    String password;
    public boolean authenticate (String a, String b){
        password = b;
        return true;
    }

    public String extractPassword(User u) {
        → u.login(this);
        → return password;
    }
}
```

takes the user's password in b & sets it to password

by calling the user's login method & passing in this, we force the user to call our authenticate method & give us the password.