Inheritance Practice

```java
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: ________________ */
        x.b();    // Output: ________________ */
        x.c();    // Output: ________________ */
        ((R)x).c(); // Output: ________________ */
        x.d();    // Output: ________________ */
        ((R)x).d(); // Output: ________________ */
    }
}
```

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.

```java
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) {
        // Your implementation here
    }

    public static void main(String[] args) {
        ArrayList<Integer> integers = new ArrayList<>();
        integers.add(2); integers.add(3); integers.add(4);
        ________ add = ____________________;
        ________ mult = ____________________;
        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:
```
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.

```java
public interface Comparator<T> {
    /** Compares its two arguments for order.
     * Returns a negative integer, zero, or a positive integer if the first
     * argument is less than, equal to, or greater than the second. */
    int compare(T o1, T o2);

    /** Indicates whether some other object is "equal to" this
     * comparator. (You don’t need to implement this for this problem) */
    boolean equals(Object obj);
}
```

```java
import java.util.ArrayList;
import java.util.Collections;
public class Animal {
    private String name;
    private int age;
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }
    /** Returns this animal’s age. */
    public int getAge() {
        return this.age;
    }
    public static void main(String[] args) {
        ArrayList<Animal> animals = new ArrayList<>();
        animals.add(new Animal("Garfield", 4));
        animals.add(new Animal("Biscuit", 2));
        AnimalComparator c = new AnimalComparator(); //Initialize comparator
        Collections.sort(animals, c);
    }
}
```

```java
import java.util.Comparator;
public class AnimalComparator implements Comparator<Animal> {
    
}
```
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 ________________ {

    public String extractPassword(User u) {

    }
}