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);
    }
    static void run(Q x) {
        x.a();
        x.b();
        x.c();
        ((R)x).c();
        x.d();
        ((R)x).d();
    }
}

Write next to each line in run what it prints.
2 Reduce

We’d like to write a method `reduce`, which uses a binary function to accumulate the values of a `List` of integers into a single value. `reduce` will need to take in an object that can operate (through a method) on two integer arguments and return a single integer. Note that `reduce` must work with a range of binary functions (addition and multiplication, for example). 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 {
    /** Apply a function of two arguments cumulatively to the
     * elements of list and return a single accumulated value. */
    static int reduce( ___________ func, List<Integer> list) {

    }

    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
    }
}

//Add additional classes and interfaces below:
```
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
import java.util.ArrayList;
import java.util.Collections;

public class Animal {
  protected String name, noise;
  protected int age;
  public Animal(String name, int age) {
    this.name = name;
    this.age = age;
    this.noise = "Huh??";
  }
  public int getAge() {
    return this.age;
  }
}

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

```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. */
  boolean equals(Object obj);
}
```

```java
public class Animal {
  protected String name, noise;
  protected int age;
  public Animal(String name, int age) {
    this.name = name;
    this.age = age;
    this.noise = "Huh??";
  }
  public int getAge() {
    return this.age;
  }
}
```

```java
import java.util.ArrayList;
import java.util.Collections;
public class Animal {
  protected String name, noise;
  protected int age;
  public Animal(String name, int age) {
    this.name = name;
    this.age = age;
    this.noise = "Huh??";
  }
  public int getAge() {
    return this.age;
  }
}
```

```java
public class AnimalComparator implements Comparator< ______ > {  
}
```
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.