1 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();
        x.b();
        x.c();
        ((R)x).c();
        x.d();
        ((R)x).d();
    }
}
```

In `run`, write what gets printed next to each line.
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) {

    }

    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.

Note: for this question, you do not need to worry about implementing equals.

```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
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));
        Animal Comparator c = new AnimalComparator(); //Initialize comparator
        Collections.sort(animals, c);
    }
}
```

```java
import java.util.Comparator;
public class AnimalComparator implements Comparator<_______________> {
}
```
public class PasswordChecker {
    /**
     * Asks you to login (by providing your username and password)
     */
    public void loginPrompt(User u) {
        u.login(this);
    }
    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) {

    }
}
5  A Bit on Bits

Complete the following two functions.

/**
 * Returns whether the ith bit of num is a 1 or not. i = 0 represents the least
 * significant bit, i = 1 represents the bit to the left of that, and so on.
 * For example, if num = 2, then i = 0 for it is not on but i = 1 is on since 2
 * in binary is 10.
 */
public static boolean isBitOn(int num, int i) {
    int mask = 1 ________________________________;
    return ____________________________________;
}

/**
 * Returns the input number but with its ith bit changed to a 1. Again, i = 0
 * represents the least significant bit, i = 1 represents the bit to the
 * left of that, and so on.
 * For example, if num = 1, which in binary is 01, then turning its i = 1 bit on
 * would result in the binary number 11, which is 3.
 */
public static boolean turnBitOn(int num, int i) {
    int mask = 1 ________________________________;
    return ____________________________________;
}