Review practice

Part 1 Boxes and Pointers

Problem 1

```java
public class Car {
    private String model;
    private String color;

    public Car(String model, String color) {
        this.model = model;
        this.color = color;
    }

    public void repaint(String newColor) {
        this.color = newColor;
    }

    public String getModel() {
        return model;
    }

    public String getColor() {
        return color;
    }
}
```

```java
public class Garage {
    private Car[] cars;
    private int nextIndex;

    public Garage(int capacity) {
        cars = new Car[capacity];
        nextIndex = 0;
    }

    public void parkCar(Car car) {
        if (nextIndex < cars.length) {
            cars[nextIndex] = car;
            nextIndex++;
        } else {
            System.out.println("No more space in the garage!");
        }
    }

    public Car[] getCars() {
        return cars;
    }
}
```

Q1

```java
public static void main(String[] args) {
    Garage myGarage = new Garage(2);

    Car car1 = new Car("Mustang", "Red");
    Car car2 = new Car("Civic", "Blue");

    myGarage.parkCar(car1);
    myGarage.parkCar(car2);

    // Repaint car2
    car2.repaint("Green");
}
```

Q2

```java
public static void main(String[] args) {

    Car[] cars = new Car[3];

    Car car1 = new Car("Mustang", "Red");
    Car car2 = new Car("Civic", "Blue");

    cars[0] = car1;
    cars[1] = car2;

    car1 = car2;
    car1.repaint("Green");
}
```

Problem 2 (more challenging)

```java
public class Egg {
    private int sizeInCm;
    private String color;

    public Egg(int size, String color) {
        this.sizeInCm = size;
        this.color = color;
    }

    public void paintEgg(String newColor) {
        color = newColor;
    }

    public String getColor() {
        return this.color;
    }
}
```

```java
public class Nest {
    private Egg[] myEggs;
    private int nextIndex;

    public Nest(int capacity) {
        this.myEggs = new Egg[capacity];
        this.nextIndex = 0;
    }

    public Nest(double height, Egg[] presents, int nextIndex, int max) {
        this.myEggs = presents;
        this.nextIndex = nextIndex;
    }

    public Egg stealColoredEgg( String colorToSteal) {
        for ( int i=0; i< myEggs.length; i++) {
            if (this.myEggs[i].getColor().equals(colorToSteal)) {
                Egg stolenEgg = this.myEggs[i];
                this.myEggs[i] = null;
                return stolenEgg;
            }
        }
        return null;
    }

    public void addEgg(Egg e) {
        this.myEggs[this.nextIndex] = e;
        this.nextIndex += 1;
        if (this.nextIndex == myEggs.length) {
            this.nextIndex= 0;
        }
    }
}
```

Q1

```java
public static void main(String[] args) {
    Nest nest1 = new Nest(2);
    Egg egg1 = new Egg(4, "Blue");
    nest1.addEgg(egg1);
    egg1.paintEgg("Green");
}
```

Q2

```java
public static void main(String[] args) {
    Egg[] eggs = new Egg[3];
    eggs[0] = new Egg(2, "Yellow");
    eggs[1] = new Egg(4, "Red");

    Nest nest = new Nest(3);
    nest.addEgg(eggs[0]);
    nest.addEgg(eggs[0]);
    nest.addEgg(eggs[1]);
    nest.addEgg(eggs[1]);

    eggs[2] = nest.stealColoredEgg("Yellow");
}
```

Part 2 Questions

Q1. Which of the following is a valid interface declaration in Java?

- **A)** `public interface Driveable { void drive(); }`

- **B)** `interface Driveable { public abstract void drive() {} }`

- **C)** `public interface Driveable { default void drive() {} }`

- **D)** Both A and C

Q2. True or False: In Java, Hasmap stores elements in a sorted order

Q3. Which of the following are valid ways to declare an array of integers in Java? (Select all that apply)

- **A)** `int[] arr;`

- **B)** `int[] arr = new int[5];`

- **C)** `int arr = new int[5];`

Q4. Which of the following is NOT a feature of an ArrayList in Java?

- **A)** Dynamic resizing.

- **B)** Allows duplicate elements.

- **C)** Provides indexed access.

- **D)** Fixed size.

Q5. True or False: An ArrayList in Java can only store objects, not primitive data types.

Q6. Which of the following are characteristics of encapsulation? (Select all that apply)

- **A)** Encapsulation hides the internal state of an object.

- **B)** Encapsulation allows direct access to object fields from outside the class.

- **C)** Encapsulation is achieved using private fields and public getter/setter methods.

- **D)** Encapsulation leads to increased coupling.

Q7. Which of the following is the best design principle?

- **A)** Low coupling / High cohesion.

- **B)** Low coupling / Low cohesion.

- **C)** High coupling / Low cohesion.

- **D)** High coupling / High cohesion

Q8 From the code provided, what is the output?

```
HashMap<String, Integer> map = new HashMap<>();
map.put("apple", 3);
map.put("banana", 2);
map.put("apple", 5);
System.out.println(map.get("apple"));
```

Answer:

Q9. Which of the following statements best describes coupling in software design?

- **A)** The degree to which a system's components depend on each other.

- **B)** The ability to reuse code across multiple classes.

- **C)** The distribution of responsibilities across different modules.

- **D)** The speed at which the program executes.

Q10. High cohesion in a class means:

- **A)** The class has many responsibilities.

- **B)** The class is highly interconnected with other classes.

- **C)** The class's methods and fields are closely related to a single task.

- **D)** The class has a lot of global variables.

Q11. Which of the following best exemplifies the message chain (that we should avoid)?

- **A)** A class that does not call any methods on other objects.

- **B)** Method calls like `a.getB().getC().doSomething()`.

- **C)** A class with many getters and setters.

Q12. Which of the following are characteristics of dependency in object-oriented design? (Select all that apply)

- **A)** A class relying on another class for its functionality.

- **B)** A change in one class can potentially affect another class.

- **C)** Dependencies can always be resolved by making all classes public.

- **D)** Dependencies can be minimized using design patterns.

Q13. True or False: An interface in Java can contain instance variables.

Q14. What is the output

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);
list.remove(1);
System.out.println(list.get(1));
```