A

Java Puzzle Ball Challenge Questions Answered

Topics

- Lesson 6
- Lesson 8
- Lesson 10
- Lesson 11
- Lesson 12

Question 1

How many objects can you identify in the game?

- Red wheels
- Blue wheels
- Ball
- Duke

ORACLE

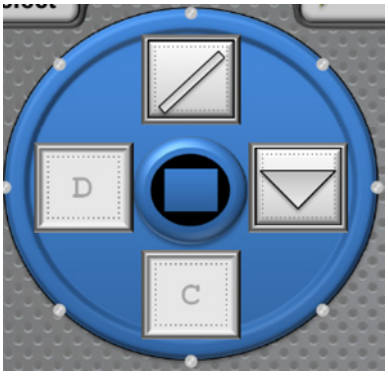
Discussion

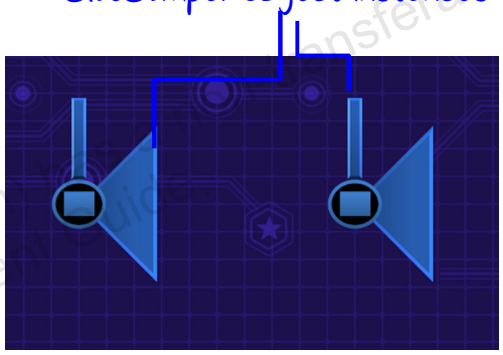
You might also have answered that the walls are objects. Remember that objects are not always physical. For instance, you might suggest that Score is an object, or Level (the level of the game). The business of identifying objects for an object-oriented application is more art than science.

Question 2

Given that a class is a blueprint for an object, which game components best reflect the class/instance relationship?

- The blue objects are *instances* of the BlueWheel class.
- The objects share the properties and methods of the BlueWheel class.





Discussion

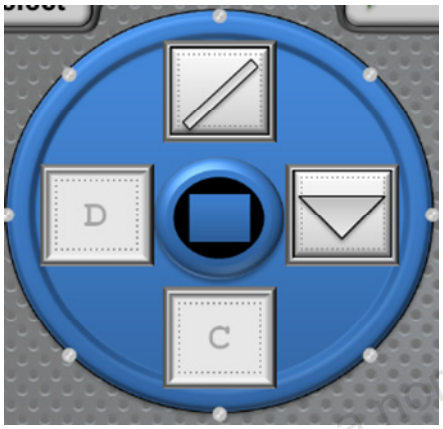
Here is an example of one of the game's objects, the blue bumper, and its function in the game. BlueBumper is the class, and a class is a blueprint or recipe for an object. The class describes an object's properties and behaviors. Classes are used to create object instances, such as the two BlueBumper object instances, as shown in the second image.

Initially, each object instance looks the same, but as you noticed, objects can change and differentiate themselves as play continues.

Question 3

How many object properties can you find?

- Color
- Shape
- Orientation
- X position
- Y position

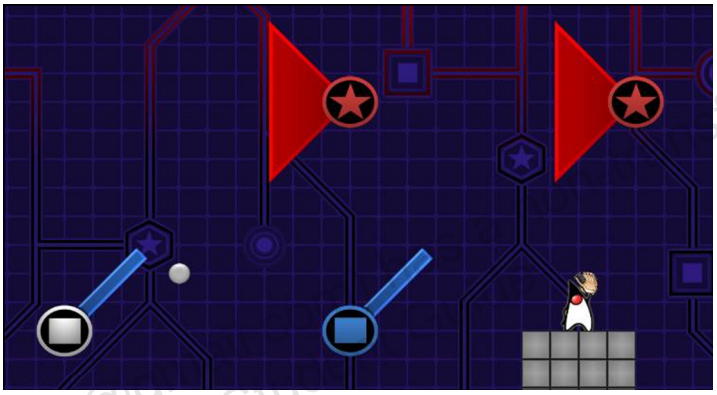


There are many others you could have named as well.

Question 4

Can you guess what some of the methods might be?

- Behaviors:
  - divertCourseSimple
  - divertCourseTriangle
  - rotate
  - play



There are many possibilities of methods for this game. Methods are actions that occur in a program. Here are a few of the possible methods. You will, no doubt, think of many more.

- You know that when the ball strikes a wall, its course will be diverted, but the change in course will be different if the wall is a simple wall or if it is a triangle wall. Consequently, you assume that these are two different methods.
- You also know that when you rotate one of the wheels, the objects of the same color are also rotated, so you can assume that there must be a rotate method.
- When you click Play, the ball starts to move, so you can assume that a play method must exist.

Note: When you play the other puzzles of the game, you will see several other types of walls and different behaviors.

Topics

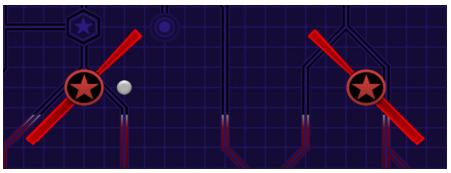
- Lesson 6
- Lesson 8
- Lesson 10
- Lesson 11
- Lesson 12

Question 1

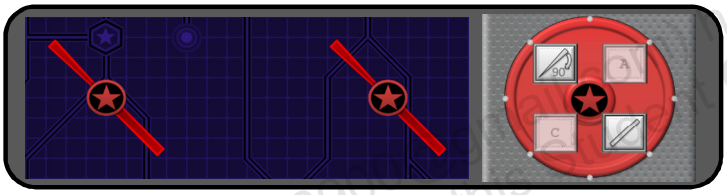
Which of the scenarios below reflects the behavior of:

- A static variable?
- An instance variable?

1. A single bumper rotates after being struck by the ball.



2. Rotating the red wheel changes the orientation of all red bumpers.



ORACLE

In Scenario 1, you see the behavior of an instance variable. When the rotation wall is struck, only that particular object changes its orientation. Other objects retain their previous orientation. This is the behavior you would expect to see with an instance variable.

In Scenario 2, you see the behavior of a static variable exhibited. When you change the orientation of a wheel, all objects of that same color are also rotated and, therefore, share the same orientation as the wheel.

Topics

- Lesson 6
- Lesson 8
- Lesson 10
- Lesson 11
- Lesson 12

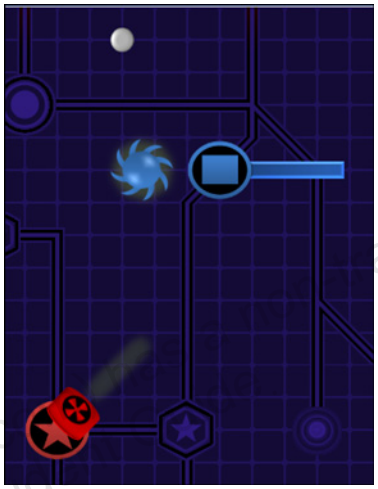
Question 1

- What type of conditional construct would you use to handle the behavior of the blade?

  Option 1: Using an `if/else` construct

  – Pseudocode example

```
If object struck != fan
    destroy object struck
    change blade to a ball
else
    divert course
    destroy the next object struck
```



As you saw in the game, there is some fairly complex logic implemented that determines both the trajectory of the ball and also the properties of the objects it strikes. When the appearance of the ball has changed to a blade, it destroys the object it strikes—but not if the object is a fan.

- There are several ways you could structure this logic. You will examine two options here.
- On this page, you see a pseudocode example of how you might handle the conditional logic using and `if/else` construct. It first tests for the exception (that is, not destroying the object) and then tests for all remaining conditions.

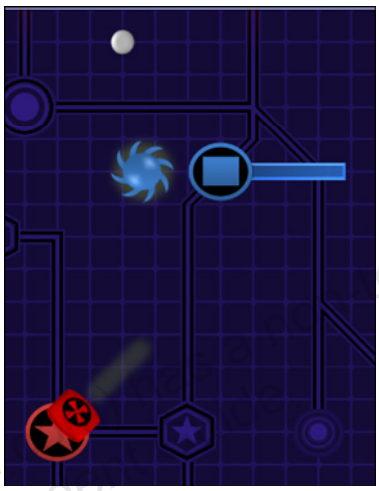
Question 1

- What type of conditional construct would you use to handle the behavior of the blade?

  Option 2: Using a `switch` construct

  – Pseudocode example

```
switch objectStruck
    case wall or rotation or triangle
       destroy objectStruck
        change blade to a ball
        break
     case fan
        divert course
        destroy the next object
```



ORACLE

- Here you see an example using a `switch` construct. You will notice that this logic is structured similarly to the `if/else` construct. The main difference is that it reverses the conditional test. It first tests all of the cases in which the object *should* be destroyed, then tests for the exception.
- Note that, because a switch statement can only evaluate a single value of type `int`, `short`, `byte`, `char`, or `String`, you would have to evaluate the object struck using some derived value (an object ID for instance).

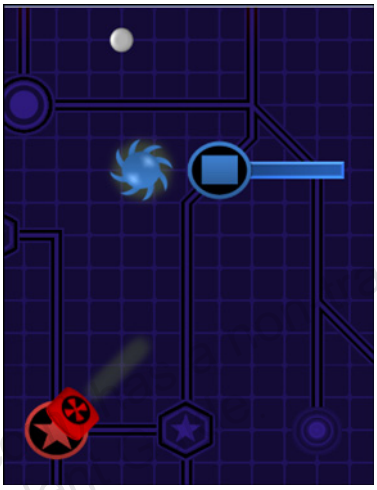
Topics

- Lesson 6
- Lesson 8
- Lesson 10
- Lesson 11
- Lesson 12

Question 1

- How might you structure the logic of the blade behavior using a `while` loop?
  - Pseudocode example

```
While ball image is “blade”
    if object struck == fan
       continue loop
    else
       destroy object struck
       change ball image to “ball”
```



In the pseudocode example, a `while` loop is executed as long as the image representing the ball is a blade. It continues to the next loop iteration if it encounters a fan. Thus, the loop will continue to iterate until it strikes something other than a fan. When this happens, the ball ceases to be a blade, so the loop exits.

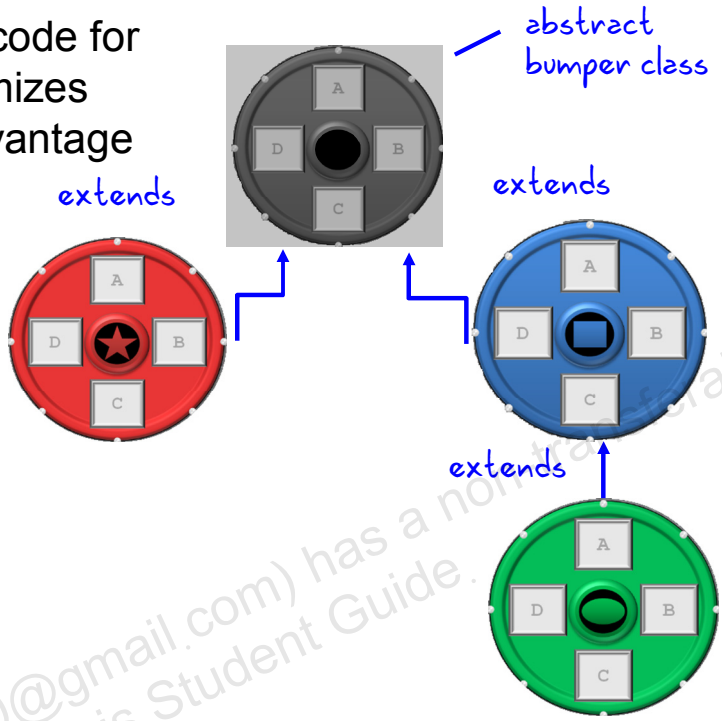
You might have used a different approach to the `while` loop. Again, there are many approaches that would work.

Topics

- Lesson 6
- Lesson 8
- Lesson 10
- Lesson 11
- Lesson 12

Question 1

Is there a way to design code for these bumpers that minimizes duplication and takes advantage of polymorphism?



In the basic puzzles, you saw a red wheel and a blue bumper. In the inheritance puzzles of the game, green bumpers were introduced. You saw that the green bumper seems to inherit from (extend) the blue bumper. Obviously, all three of these bumpers share functionality and properties.

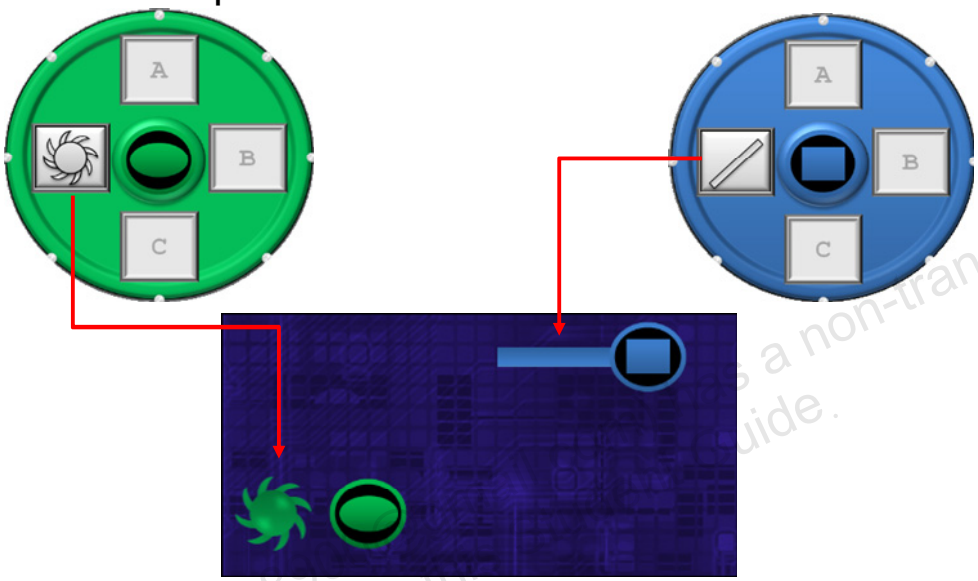
An abstract class can be used to impose the interface that we see in all of these classes. It might also include a concrete method or property.

The graphic above shows this modified hierarchy (Remember from the lesson titled “Describing Objects and Classes” how an instance was best represented by a bumper, and a class was best represented by the wheel). Assuming that the green bumper class extends the blue bumper class, it derives all of its behaviors and properties from the abstract bumper class through the blue bumper class. Only red and blue bumper classes directly extend the abstract bumper class.

Question 2

To make overriding possible, which game components best represent:

- A method name and signature?
- A method implementation?



When you drag an icon, such as the simple wall in the example above, to a slot in the blue wheel (slot D), all green objects show the simple wall in the D position as well.

If, however, you then drop a different icon (blade, in this example) to the D slot of the green wheel, the simple wall is replaced by the blade. The icon would therefore represent a new implementation of the method found in slot D. You could consider slot D to represent a method name and signature, such as `slot_D()` or `D_method()`.