Announcements

Assignment 3

In this assignment you will develop a class to work with solos in jMusic.

DEADLINE
This assignment is due on Wednesday, December 11 at 11:00 pm.

DESCRIPTION
A LITTLE MUSIC MATH

Music is very mathematical when you break it down. Don't worry if you don't have a music background. This section should bring you up to speed on what you need to know for the next part.

The key of a song describes which pitches fit into a song, and which do not. A key can be described by a root and a scale. The root is an integer value pitch that determines the start point of the scale. The scale is an array of ints that determines which offset pitches are "in-key."

Let's start with the major scale. The int array for the major scale is

\{0, 2, 4, 5, 7, 9, 11\}.

This is also given by JMC\_MAJOR\_SCALE.

Now, given any root \(N\), figuring out which pitches are in the key is just a simple math problem. The following pitches are in the key:

\(N, N+2, N+4, N+5, N+7, N+9, N+11\)

Which means, of course that the following pitches are not in the key:

\(N+1, N+3, N+6, N+8, N+10\).

Take the root of C for example. Using C4 \(= 60\) as an example, the following pitches are in the key of C Major:

60, 62, 64, 65, 67, 69, 71

Also, C5 \(= 72\), so the following pitches are also in the key of C Major:

72, 74, 76, 77, 79, 81, 83

In fact, adding or subtracting 12 to any root gives the same note letter in a different octave, so we can use this formula to determine if any pitch \(P\) \((0 \leq P \leq 127)\) is in the key or not.

Furthermore, this can be done with any root \(R\) \((0 \leq R \leq 127)\), and any scale given as an array of ints \(S = \{S_1, S_2, ..., S_n\}\) where \(0 \leq S_i \leq 11\) for \(0 \leq i \leq n\).
ICLICKER QUESTION

Overloaded methods are

A. Methods with an inefficient time complexity
B. Methods with an inefficient space complexity
C. Multiple methods with the same name
D. Methods with more than one statement
E. Methods that call themselves

Principles of OO Programming

- Encapsulation
  - Objects can combine data and operations
- Inheritance
  - Classes can inherit properties from other classes
- Polymorphism
  - Objects can determine appropriate operations at execution time

Inheritance

- Inheritance
  - Allows a class to derive the behavior and structure of an existing class
Inheritance

- Superclass or base class
  - A class from which another class is derived
- Subclass, derived class, or descendant class
  - A class that inherits the members of another class
- Benefits of inheritance
  - It enables the reuse of existing classes
  - It reduces the effort necessary to add features to an existing object

A subclass
- Can add new members to those it inherits
- Can override an inherited method of its superclass
  - A method in a subclass overrides a method in the superclass if the two methods have the same declarations

Circle
- radius
- area()
- toString()

Cylinder
- radius
- height
- area()
- volume()
- toString()
Inheritance

public class Circle
{
    double radius;

    public Circle(double radius)
    {
        this.radius = radius;
    }

    public double area()
    {
        return Math.PI * radius * radius;
    }

    public String toString()
    {
        return "A round object with area = " + area();
    }
}

public class Cylinder extends Circle
{
    double height;

    public Cylinder(double radius, double height)
    {
        super(radius);
        this.height = height;
    }

    public double area()
    {
        return 2.0 * Math.PI * radius * (radius + height);
    }

    public double volume()
    {
        return Math.PI * radius * radius * height;
    }
}

public class Sphere extends Circle
{
    public Sphere(double radius)
    {
        super(radius);
    }

    public double area()
    {
        return 4.0 * Math.PI * radius * radius;
    }

    public double volume()
    {
        return (4.0 / 3.0) * Math.PI * radius * radius * radius;
    }
}
public class Sphere extends Circle {

    public Sphere(double radius) {
        super(radius);
    }

    public double area() {
        return 4.0 * super.area();
    }

    public double volume() {
        return (4.0 / 3.0) * Math.PI * radius * radius * radius;
    }
}

Inheritance

- A subclass inherits private members from the superclass, but cannot access them directly
- Methods of a subclass can call the superclass’s public methods
- Clients of a subclass can invoke the superclass’s public methods
- An overridden method
  - Instances of the subclass will use the new method
  - Instances of the superclass will use the original method

Java Access Modifiers

- Membership categories of a class
  - Public members can be used by anyone
  - Members declared without an access modifier (the default) are available to
    - Methods of the class
    - Methods of other classes in the same package
  - Private members can be used only by methods of the class
  - Protected members can be used only by
    - Methods of the class
    - Methods of other classes in the same package
    - Methods of the subclass

Access to public, protected, package access, and private members of a class by a client and a subclass
Relationships

- Two basic kinds of relationships
  - Is-a relationship
  - Has-a relationship

Is-a Relationship

- Inheritance should imply an is-a relationship between the superclass and the subclass
- Example:
  - If the class Ball is derived from the class Sphere
    - A ball is a sphere

Has-a Relationships

- Object type compatibility
  - An instance of a subclass can be used instead of an instance of the superclass, but not the other way around
Has-a Relationships

- Has-a relationship
  - Also called containment
  - Cannot be implemented using inheritance
    - Example: To implement the has-a relationship between a pen and a ball
      - Define a data field `point` whose type is `Ball` within the class `Pen`

Dynamic Binding and Abstract Classes

- A polymorphic method
  - A method that has multiple meanings
  - Created when a subclass overrides a method of the superclass
- Late binding or dynamic binding
  - The appropriate version of a polymorphic method is decided at execution time

Dynamic Binding and Abstract Classes

- Controlling whether a subclass can override a superclass method
  - Field modifier `final`
    - Prevents a method from being overridden by a subclass
  - Field modifier `abstract`
    - Requires the subclass to override the method
- Early binding or static binding
  - The appropriate version of a method is decided at compilation time
  - Used by methods that are `final` or `static`

Dynamic Binding and Abstract Classes

- Overloading methods
  - To overload a method is to define another method with the same name but with a different set of parameters
  - The arguments in each version of an overloaded method determine which version of the method will be used