Classes and Enumerations

• Assignment 3 - example of working with a class interface
• Class variables and methods
• Converting object references
• Extending classes
• Enumerations
  – Dealing with collections of objects
Assignment 3 - Example

• Define `boolean pointOnSegment(Point)` in the Segment class

• How to design this method?

Is (x3,y3) on the Segment?
Example

- Some assumptions:
  - \((x_1,y_1) \neq (x_2,y_2)\)
  - if the segment is part of a vertical line, we’ll have to use a special case, because of the properties of the \(y = \text{slope} \times x + y_{\text{intercept}}\) eqn
  - also, the following can occur in combination:
    - \(y_1 \leq y_2 \text{ OR } y_1 > y_2\) combined with
    - \(x_1 \leq x_2 \text{ OR } x_1 > x_2\).
  - after we check that the point is on the line, we must check that point is actually on the Segment
Example

- If segment is vertical, then
  - abs(x1-x2) < tolerance
    - to be on same line, then abs(x3-x1) < tolerance && y1 <= y3 <= y2 OR y2 <= y3 <= y1.

- If segment is not vertical, then need to calculate equation of line containing segment and make sure (x3,y3) is on that line between the other two points.
Equation of a Line

\[ y = m \times x + b \]
\[ y_1 = m \times x_1 + b \text{ and } y_2 = m \times x_2 + b \]

therefore, \[ m = \frac{(y_1 - y_2)}{(x_1 - x_2)}. \]

substituting that into the equation for \((x_1, y_1)\)
we derive: \[ b = y_1 - \frac{(y_1 - y_2)}{(x_1 - x_2)} \times x_1 \]
so we have,

\[ y = \frac{(y_1 - y_2)}{(x_1 - x_2)} \times x + \frac{y_1 - (y_1 - y_2)}{(x_1 - x_2)} \times x_1 \]
\[ y = m \times x + b \]
pointOnSegment(Point p)

public boolean pointOnSegment(Point p) {
// first get endpoints of Segment
double x1 = (this.getFirstPoint()).getX(),
    y1 = (this.getFirstPoint()).getY(),
    x2 = (this.getSecondPoint()).getX(),
    y2 = (this.getSecondPoint()).getY(),
    x3 = p.getX(),
    y3 = p.getY();

// calculate slope and intercept
double ptTol = Point.getTolerance();
// check for vertical Segment
if (Math.abs(x1-x2) < ptTol) {
    if (!Math.abs(x1-x3)<ptTol))
        return false;
    else {if(((y1<=y3)&&(y3<=y2)) ||
                ((y2<=y3)&&(y3<=y1)))
        return true;
    else return false;}
}
```java
//if reach here, Segment is not vertical
//have to check that (x3,y3) is on same line and
//between the endpoints
double m = (y1-y2)/(x1-x2),
    b = y1 - ((y1-y2)/(x1-x2))*x1;
// is (x3,y3) on line?
if (!(Math.abs(y3 - (m*x3+b)) < ptTol)) return false;

// is (x3,y3) between (x1,y1) and (x2,y2)?
if (((x1<=x3)&&(x3<=x2)) || ((x2<=x3)&&(x3<=x1)))
    return true;
else return false; //on line but outside Segment
}```
Class Variables

- Used to keep information shared by all objects created of this class
- Only one copy of a class variable; every object created in the class has access to the SAME class variable
- Static keyword denotes that a variable is a class variable rather than an instance variable

```java
public static int count;
```
Class Variables

• For example,
  – count the number of times of day which were converted by users of our UStime class

```java
public static int usecount;
```

– tolerance class variables in Point class

```java
// separation required for points to be different
private static double tolerance = 1.e-10; // tolerance for class
```
Class Methods

- Usually used to manipulate class variables or to provide functionality not linked to a particular object
  - `getTolerance()`, `setTolerance()` in Point
- Defined with keyword `static`
  
  ```
  public static void setTolerance(double t)
  public static boolean allConcentric(Set s)
  ```

- Accessed as `<classname>`.`<method-name>`
Instance versus Class Methods

• How to tell if a method should be a class method or instance method?
  – Ask if it is an action with respect to a particular object?
    – If yes, then define an instance method
    – If no, then define a class method
  – Ask if you really need this method? always a good question -:)
  – Might this method naturally be defined in another class?
Conversion between Classes

• Can always go up the tree
  – a Cat is a Carnivore
• Same idea as widening
• Always works

Cat c;
Carnivore k;
Cat c = new Cat;
k = c; // now k refers to a Cat object which is also a Carnivore.
Conversion between Classes - 2

- Moving down the hierarchy doesn’t always work; when it does, it needs a cast.
  - Some Carnivores are Cats
  - If cast fails, it generates a ClassCastException at run-time

```java
Cat c;
Carnivore k;
...
c = (Cat)k; // only works when k actually refers to a Cat object
           // which is a Carnivore.
```
An Enumeration

- Provides way to iterate over a collection of objects
- Returns each element in the collection, one by one, in no particular order
- Has two required methods
  - `hasMoreElements()`: returns `true` if enumeration has more elements to return, else returns `false`
  - `nextElement()`: returns next element of the enumeration defined to be of type `Object`
An Enumeration

- Usage often requires casting of the returned object to the proper type
- Enumeration is a Java interface, something which requires a class be created to implement the methods specified
- Can think of interface as a class which contains only specifications of methods, no implementations
- A class which implements an interface promises to define the specific set of functions in that interface
Example

- A Polygon is a set of Segment objects
- `getEdges()` in class Polygon returns an enumeration object for the constituent edges of its Polygon receiver
- Use of the enumerator, assuming Polygon `polyp` exists:

```java
Enumeration edgeEnum = polyp.getEdges();
while (edgeEnum.hasMoreElements()) {//note cast
    Segment seg = (Segment) edgeEnum.nextElement();
    System.out.println(" "+ seg.toString());
}
```
An enumeration would return segments: p1-p2, p2-p3, p3-p4, p4-p5, p5-p1, although not necessarily in that order.

After 5 `nextElement()` calls, the enumeration would be finished having returned each edge ONCE! To loop through the edges again would require a NEW Enumeration object.
What’s important?

- How to use Enumeration objects?
- Remember that if you change the underlying collection while you are enumerating over it, unforeseen results will happen
- Once you iterate over all the objects the enumeration will be used up
Perimeter of a Polygon

- First, need to calculate the length of a Segment
- Second, need to enumerate all the Segments which form the Polygon and sum their lengths
public double getLength(){
//uses distanceTo() in Point to calculate
//length of a Segment
    return
        (this.getFirstPoint()).distanceTo(
            this.getSecondPoint());
}
Perimeter of a Polygon

//method to use an enumerator to calculate
//the perimeter of a Polygon object
//needs to use getLength() from Segment class
public double getPerimeter()
{
    double perimeter = 0.0;
    Enumeration edgeEnum = this.getEdges();
    while (edgeEnum.hasMoreElements())
    {
        Segment seg = (Segment)edgeEnum.nextElement();
        perimeter += seg.getLength();
    }
    return perimeter;
}
Extending Classes - Alternative

- Could use one **tolerance** over all geometric classes in their **equals()** methods. Then could create `GeomObject` class and make all other geometric classes subclasses of it.

```
Object
  GeomObject
    tolerance - class var
    getTolerance() - inherited class method

Point
Segment
Circle
Polygon
```
Extending Classes - Alternative

- Invoke by GeomObject.getTolerance()
- Could also put some instance variables and instance methods shared by all GeomObjects in the superclass
  - color - all GeomObjects have a color
  - getColor() - an observer method
  - Point p; Color c = p.getColor()
- This class hierarchy is NOT in Assignment 3, but it represents an alternative to what we have done.