CSE 142 Section Handout #8
Cheat Sheet

Field (8.1) *(data inside each object)*

```
private type name;
```

Method (8.2) *(behavior inside each object)*

```
public type name(parameters) {
    statements;
}
```

Constructor (8.3) *(code to initialize new objects)*

```
public className(parameters) {
    statements;
}
```

Client Program *(a program that uses objects)*

```
public class PointMain {
    public static void main(String[] args) {
        Point p1 = new Point(5, -2);
        Point p2 = new Point(14, 6);

        p1.translate(4, 8);
        p2.setLocation(20, 30);

        double d = p1.distance(p2);
        System.out.println(p1 + " - " +
                p2 + " = " + d);
    }
}
```

`toString` method (8.2) *(called when an object is printed)*

```
public String toString() {
    code that produces/returns a String;
}
```

Example complete class:

```
import java.awt.*;    // for Graphics

// A Point object represents a location on the (x, y) plane.
public class Point {
    private int x;
    private int y;

    // Constructs a new Point at the origin, (0, 0).
    public Point() {
        this(0, 0);
    }

    // Constructs a new Point object at the given x/y location.
    public Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }

    // Returns the distance between this point and the given other Point p2.
    public double distance(Point p2) {
        int dx = x - p2.x;
        int dy = y - p2.y;
        return Math.sqrt(dx * dx + dy * dy);
    }

    // Draws this Point on a DrawingPanel.
    public void draw(Graphics g) {
        g.fillRect(x, y, 3, 3);
        g.drawString(toString(), x, y);
    }

    // Returns the x-coordinate of this Point.
    public int getX() {
        return x;
    }

    // Returns the y-coordinate of this Point.
    public int getY() {
        return y;
    }

    // Shifts this Point's x/y position by the given amounts.
    public void translate(int dx, int dy) {
        x += dx;
        y += dy;
    }

    // Returns a String representation of this Point, such as "(5, 18)".
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}
```

CSE 142 Section Handout #8
Questions

Classes

1. Define a class named `TimeSpan`. A `TimeSpan` object stores a span of time in hours and minutes (for example, the time span between 8:00am and 10:30am is 2 hours, 30 minutes). Each `TimeSpan` object should have the following public methods:

`TimeSpan(`**hours**, **minutes**`)`
Constructs a TimeSpan object storing the given time span of minutes and seconds.

`add(`**hours**, **minutes**`)`
Adds the given amount of time to the span. For example, (2 hours, 15 min) + (1 hour, 45 min) = (4 hours). Assume that the parameters are valid: the hours are non-negative, and the minutes are between 0 and 59.

`add(`**timespan**`)`
Adds the given amount of time (stored as a time span) to the current time span.

`getTotalHours()`
Returns the total time in this time span as the real number of hours, such as `9.75` for (9 hours, 45 min).

`toString()`
Returns a string representation of the time span of hours and minutes, such as `"28h46m"`.

The minutes should always be reported as being in the range of 0 to 59. That means that you may have to "carry" 60 minutes into a full hour. For example, consider the following client code:

```
public class TimeSpanClient {
    public static void main(String[] args) {
        TimeSpan t1 = new TimeSpan(3, 45);
        System.out.println(t1 + " is " + t1.getTotalHours() + " hours");

        t1.add(2, 30);
        System.out.println(t1 + " is " + t1.getTotalHours() + " hours");

        TimeSpan t2 = new TimeSpan(0, 51);
        t1.add(t2);
        System.out.println(t1 + " is " + t1.getTotalHours() + " hours");
    }
}
```

This code creates a `TimeSpan` object and adds three different times to it. The output should be:

```
3h45m is 3.75 hours
6h15m is 6.25 hours
7h6m is 7.1 hours
```

Notice that the second time is *not* 5 hours, 75 min, although that's what you'd get by just adding field values.

(continued on back page)

CSE 142 Section Handout #8
Questions (continued)

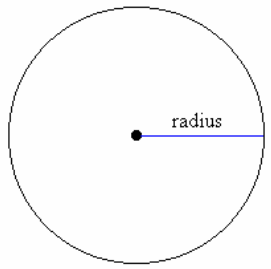
2. Define a class named `Circle`. A `Circle` object stores a radius and the (x, y) coordinates of its center point. Each `Circle` object should have the following public methods:

`Circle(`**center**`,` **radius**`)`
Constructs a new circle with a center specified by the given `Point` and with the given integer radius.

`getRadius()`
Returns the circle's radius.

`getArea()`
Returns the area occupied by the circle, using the formula πr^2.

`getCircumference()`
Returns the circle's circumference (distance around the circle), using the formula $2\pi r$.



`toString()`
Returns a string representation of the circle, such as `"Circle[center=(75, 20), radius=30]"`.

`draw(`**g**`)`
Draws the circle onto a `DrawingPanel`, labeling its center point and drawing an outline of the circle itself.

`contains(`**p**`)`
Returns whether the given point lies inside the circle.

3. Define a class named `Rectangle`. A `Rectangle` object stores an (x, y) coordinate of its top/left corner, a width, and a height. Each `Rectangle` object should have the following public methods:

`Rectangle(x, y, width, height)`
Constructs a new rectangle whose top-left corner is specified by the given coordinates and with the given width and height.

`getX()`, `getY()`, `getWidth()`, `getHeight()`
Returns the rectangle's leftmost x-coordinate, top y-coordinate, width, and height respectively.

`toString()`
Returns a String representation of the rectangle, such as `"Rectangle[x=1,y=2,w=3,h=4]"`.

`draw(`**g**`)`
Draws the rectangle onto a `DrawingPanel`, labeling its top/left and bottom/right corners.

`contains(`**p**`)`
Returns whether the given point lies inside the bounds of the rectangle.

`contains(`**rect**`)`
Returns whether the given rectangle lies entirely inside the bounds of this rectangle.

`union(`**rect**`)`
Turns this rectangle into the union of itself and the given other rectangle *rect*; that is, the smallest rectangular region that completely contains both this rectangle and the given other rectangle.

CSE 142 Section Handout #8
Solutions

1.

```
// A TimeSpan object represents a duration of time in hours and minutes.
public class TimeSpan {
    private int hours;
    private int minutes;

    // Constructs a time span representing the given number of hours and minutes.
    public TimeSpan(int initialHours, int initialMinutes) {
        hours = 0;
        minutes = 0;
        add(initialHours, initialMinutes);
    }

    // Adds the given hours/minutes to this time span, wrapping hours if necessary.
    public void add(int initialHours, int initialMinutes) {
        hours += initialHours;
        minutes += initialMinutes;
        if (minutes >= 60) {
            minutes -= 60;   // convert 60 min --> 1 hour
            hours++;
        }
    }

    // Adds the given hours/minutes to this time span, wrapping hours if necessary.
    public void add(TimeSpan span) {
        add(span.hours, span.minutes);
    }

    // Returns the total hours represented by this time span,
    // such as 7.75 for 7 hours, 45 minutes.
    public double getTotalHours() {
        return hours + minutes / 60.0;
    }

    // Returns a text representation of this time span, such as "7h45m".
    public String toString() {
        return hours + "h" + minutes + "m";
    }
}
```

2.

```
import java.awt.*;   // for Graphics

// Each Circle object represents a circle in the 2D plane
// with a given center and radius.
public class Circle {
    private Point center;   // fields
    private int radius;

    // constructor(s)
    public Circle(Point initialCenter, int initialRadius) {
        center = initialCenter;
        radius = initialRadius;
    }

    // Returns the radius of the circle.
    public int getRadius() {
        return radius;
    }

    // Returns the area of this circle.
    public double getArea() {
        return Math.PI * Math.pow(radius, 2);
    }

    // Returns the circumference of this circle (distance around the circle).
    public double getCircumference() {
        return 2 * Math.PI * radius;
    }

    // Returns whether the given point lies inside this circle.
    public boolean contains(Point p) {
        return center.distance(p) <= radius;
    }

    // Returns a text representation of this circle, such as
    // "Circle{center=(40, 100),radius=100}".
    public String toString() {
        return "Circle[center=" + center + ",radius=" + radius + "]";
    }
```

```
    // Draws this Circle onto a DrawingPanel.
    public void draw(Graphics g) {
        g.drawOval(center.getX() - radius, center.getY() - radius,
                   2 * radius, 2 * radius);
        center.draw(g);
    }
}
```

3.

```
import java.awt.*;   // for Graphics

// Each Rectangle object represents a 2D rectangle with a
// top-left x/y coordinate, width, and height.
public class Rectangle {
    private Point topLeft;
    private int width;
    private int height;

    // Constructs a new rectangle
    public Rectangle(Point initialTopLeft, int initialWidth, int initialHeight) {
        topLeft = initialTopLeft;
        width = initialWidth;
        height = initialHeight;
    }

    // Returns this rectangle's leftmost x coordinate.
    public int getX() {
        return topLeft.getX();
    }

    // Returns this rectangle's topmost y coordinate.
    public int getY() {
        return topLeft.getY();
    }

    // Returns this rectangle's width.
    public int getWidth() {
        return width;
    }

    // Returns this rectangle's height.
    public int getHeight() {
        return height;
    }

    // Returns a text representation of this rectangle.
    public String toString() {
        return "Rectangle[x=" + getX() + ",y=" + getY() +
               ",w=" + width + ",h=" + height + "]";
    }

    // Draws this rectangle onto a DrawingPanel.
    public void draw(Graphics g) {
        g.drawRect(getX(), getY(), width, height);
        topLeft.draw(g);
        Point bottomRight = new Point(getX() + width, getY() + height);
        bottomRight.draw(g);
    }

    // Returns whether the given point is contained within this rectangle.
    public boolean contains(Point p) {
        return getX() <= p.getX() && p.getX() < (getX() + width) &&
               getY() <= p.getY() && p.getY() < (getY() + height);
    }

    // Returns whether the given rectangle is entirely contained in this rectangle.
    public boolean contains(Rectangle r) {
        return getX() <= r.getX() && r.getX() + r.getWidth() < (getX() + width) &&
               getY() <= r.getY() && r.getY() + r.getHeight() < (getY() + height);
    }

    // Turns this rectangle into the smallest area that contains itself and rect.
    public void union(Rectangle r) {
        // find the union's bounds
        int left = Math.min(getX(), r.getX());
        int top = Math.min(getY(), r.getY());
        int right = Math.max(getX() + width, r.getX() + r.getWidth());
        int bottom = Math.max(getY() + height, r.getY() + r.getHeight());

        topLeft.setLocation(left, top);
        width = right - left;
        height = bottom - top;
    }
}
```