

The reason for sub-classes

- At many cases, the real reason for making sub-classes is for easy extensions. In this way, our program can last longer.
- We will see one such example in the next 3 hours of this course (saving the last 2 hours for revisions and Q&A).
- We build a program that asks the description of some figures and then draw them on the (text) screen.
- The set of possible figure types is not known when the program is designed. We want the program to have the extensibility that it needs little changes, if any, in order to add support for another type of figures.

0911B L25-1

Problem: Drawing application

- The program asks for the number of figures at the beginning.
- Then for each figure, the program reads the figure:
 - It asks for the type of figure.
 - It asks for depth of the figure, character to use for the figure and the location of the figure.
 - It asks for further information (e.g., width) about the figure.
- It draws all the figures. (For each point on the text screen, we determine the least deep figure that contains the point, and draw the corresponding character.)

0911B L25-2

Example input

For example, here might be what the program does:

```
Number of figures? 3
Figure type? circle
Depth? 0
Fill character? .
x? 0
y? 0
radius? 3
Figure type? circle
Depth? 5
Fill character? #
x? 0
y? 0
radius? 5
Figure type? rectangle
Depth? 3
Fill character? =
x? -3
y? 0
width? 12
height? 3
```

```
#####
#####
###...###
###...###
##.....##
##.....=====
##.....=====
##=.....=====
#####
#####
```

0911B L25-3

The extensibility problem

- The program would be a boring one if we know exactly what are all the possible types of figures.
- We would represent the figure as an object, with a field specifying the type.
- We then have a long if/else statement to choose a type of figure and create it.
- Then for each of the points on the screen, we use another long if/else statement to check whether the point is in each of the figures. If so we print that fill-character.
- But what if we don't know all the types? What if we can envision that we will later want to add support for other figures?

0911B L25-4

Coding extensibility

- It is no good to use if/else statements this way—if we extend the program, we will have to search for these statements and add support of the new figure for each.
- Instead of using a figure class to represent each object, we use a sub-class of the figure class.
- The figure class specifies that each sub-class can read an object and can check whether a point is in the object. Whenever we implement a new figure type, we make one more subclass of figure.
- The figure class is also responsible for dealing with those attributes that is common to all figures, i.e., location, depth and fill characters.

0911B L25-5

The Figure class

The figure class do the following:

- Every figure has a integer depth, integer x and y as location, and a fill character.
- A figure object **can read itself**. By default, asks the user for depth, x, y and fill character. Subclass needing more overrides it.
- A figure object knows **whether it contains a given point**. This is used to determine which figure to draw at each point on the screen. No default implementation: sub-classes **must** define it.
- A figure object can tell its fill character (just return it).
- We can compare objects. The comparison is based on the depth.

0911B L25-6

The Figure class outline

The class should look like the following:

```
abstract class Figure implements Comparable {
    // Read itself from StdIn
    public void read(chapman.io.StdIn in) { ... }
    // Check whether the figure contains (loc_x, loc_y)
    abstract boolean contains(int loc_x, int loc_y);
    // Get the fill character
    public char getFill() { ... }
    // Compare depths
    public int compareTo(Object obj) { ... }
    protected int x, y;
    private int depth;
    private char fill_char;
}
```

0911B L25-7

```
abstract class Figure implements Comparable {
    // Read itself from StdIn
    public void read(chapman.io.StdIn in) {
        System.out.print(" Depth? ");
        depth = in.readInt();
        System.out.print(" Fill character? ");
        fill_char = in.readChar();
        System.out.print(" x? ");
        x = in.readInt();
        System.out.print(" y? ");
        y = in.readInt();
    }
    // Check whether the figure contains (loc_x, loc_y)
    abstract boolean contains(int loc_x, int loc_y);
    // Get the fill character
    public char getFill() {
        return fill_char;
    }
    // Compare depths
    public int compareTo(Object obj) {
        Figure fig = (Figure) obj;
        return depth - fig.depth;
    }
    protected int x, y;
    private int depth;
    private char fill_char;
}
```

0911B L25-9

Protected data fields

Notice that we made x and y protected, so sub-classes can access it.

- Most likely we do not want any data field to be public, since then the user of the class must know the internal representation to use it.
- But then we still have the choice to make it either private or protected.
- For a class that do not want to have much guarantee about the data fields (e.g., x and y), it makes sense to make it protected, so sub-classes can access them directly.
- But we could make it "safer" by making x and y private and provide instance methods for getting them.

0911B L25-8

Circle: a concrete implementation

To actually be able to create an object, we must have a sub-class of Figure. For an easy one, let's make circle:

```
class CircleFigure extends Figure {
    public boolean contains(int loc_x, int loc_y) {
        int dx = loc_x - x;
        int dy = loc_y - y;
        return Math.round(Math.sqrt(dx * dx + dy * dy)) <= radius;
    }
    public void read(chapman.io.StdIn in) {
        super.read(in);
        System.out.print(" radius? ");
        radius = in.readInt();
    }
    private int radius;
}
```

Question: what are the data fields of a CircleFigure?

New: super.read(in) call Object.read(StdIn): "chaining call".

0911B L25-10