Runtime Organization of O-O Languages

- How to represent/implement object-oriented constructs such as objects, classes, methods, instance variables, and method invocation
- We already know quite a bit about this, but let’s bring it all together
Runtime Organization of O-O Languages

- Objects are a lot like records
- Instance variables are a lot like fields
- Classes are a lot like ???
- Methods are a lot like functions / procedures
- But, there are a few differences:
  - Methods have to know what object they are being invoked on
    - this
  - Object classes often support inheritance
A simple Java object

class Shape {
    int x, y;
    public Shape(int x, int y) {
        this.x = x; this.y = y;
    }
    public void move(int dx, int dy) {
        x = x + dx; y = y + dy;
    }
    public float area() {...}
    public float distance(Shape o) {...}
}
```java
public Shape(int x, int y) {
    this.x = x; this.y = y;
}

• becomes

Shape$Shape(Shape *this, int x, int y) {
    this->x = x; this->y = y;
}
```
public void move(int dx, int dy) {
    x = x + dx; y = y + dy;
}

• becomes

Shape$move(Shape *this, int dx, int dy) {
    this->x = this->x + dx;
    this->y = this->y + dy;
}
A simple Java object

Shape s = new Shape(2, 3);
Shape r = new Shape(0, 0);
Inheritance

• One object class can extend another one
  – Let’s restrict our attention to single inheritance for now

• What does this mean?
  – Everything that is contained in the parent class is part of the child class
  – The child can add new stuff that isn’t in the parent
  – The child can override any method that it doesn’t like from the parent
  – But can still get the parent behaviour
    • super
Shapes and Points

class Shape {
    int x, y;
    public Shape(int x, int y) {
        this.x = x; this.y = y;
    }
    public void move(int dx, int dy) {
        x = x + dx; y = y + dy;
    }
    public float area() {...}
    public float distance(Shape o) {...}
}
Shapes and Points

class Point extends Shape {
    float spin;
    public Point(int x, int y, float spin) {
        super(x, y); this.spin = spin;
    }
    public float area() {
        return 0.0f;
    }
    public void respin(float spin) {
        this.spin = spin;
    }
}
Shapes and Points

Shape `s` = new Shape(2, 3);
Shape `r` = new Point(0, 0, 3.0f);
Testing class membership

- How do we compile:

  ```java
  Shape s; Point p;
  s = ...
  if (s instanceof Point)
  p = (Point) s;
  ```

- First, what does it mean?

- Second, how do we implement it?
Testing class membership

• How do we compile:

```java
Object o; Shape s;
o = new Point(...);
if (o instanceof Shape)
s = (Shape) o;
```

• Why is this harder than the last one?
Shapes and Points

- instanceof Shape

Shape class

- move
- area
- dist

Point class

- move
- area
- dist

instanceof Shape

Shape

- S.Shape
- S.move
- S.area
- S.dist

Point

- P.Point
- S.move
- P.area
- S.dist
- P.respin

Object Oriented Languages (Chapter 14)
Shapes and Points

- instanceof Shape

- Shape class
  - Object
  - S.Shape
  - S.move
  - S.area
  - S.dist

- Point class
  - P.Point
  - S.move
  - P.area
  - S.dist
  - P.respin

- class
  - x: 2
  - y: 3

- Shape
  - move
  - area
  - dist

- Point
  - move
  - area
  - dist
  - respin

- Spin: 3.0f

Object Oriented Languages (Chapter 14)
Multiple Inheritance

• The techniques that we have talked about don’t work when we have multiple inheritance
  – Why not?
  – What can we do about it?
  – Why do we care?

• The book discusses a collection of these issues

• It is pretty tricky

• Even Java interfaces make it tricky

• I personally believe that general multiple inheritance is completely evil
  – Although I accept that Java’s interfaces are completely good!
interface Boring {
    int a(int x, float y);
    float b();
}
class Q implements Boring {
    void c();
    float b();
    void d();
    int a(int x, float y);
}
Boring b = new Q();
b.a(2, 3.14);
interface Trivial {
    void x();
}

class R implements Trivial, Boring {
    int a(int x, float y);
    void x();
    void x();
    void d();
    float b();
}

Boring b = new R();
b.a(2, 3.14);
## Enter itables

<table>
<thead>
<tr>
<th>Boring iface</th>
<th>Q Class</th>
<th>R Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>Object</td>
<td>R</td>
</tr>
<tr>
<td>b</td>
<td>Q.Q</td>
<td>R.R</td>
</tr>
<tr>
<td>c</td>
<td>Q.c</td>
<td>R.a</td>
</tr>
<tr>
<td>d</td>
<td>Q.d</td>
<td>R.x</td>
</tr>
<tr>
<td>a</td>
<td>Q.a</td>
<td>R.d</td>
</tr>
<tr>
<td>b</td>
<td>Q.b</td>
<td>R.b</td>
</tr>
</tbody>
</table>

| Trivial iface | |
|--------------| |
| a            | R/Trivial |
| b            | R/Boring  |
| x            | R.x       |

| a            | R.a       |
| b            | R.b       |