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) {...}
}

Object-Oriented Languages (Chapter 14)
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) {...}
}

Object-Oriented Languages (Chapter 14)
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);

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

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

s
- class
  - x: 2
  - y: 3

r
- class
  - x: 0
  - y: 0
  - spin: 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?
Shapes and Points

r instanceof Point

class

x 2

y 3

Shape class
S.move
S.area
S.dist

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

Object-Oriented Languages (Chapter 14)
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

oi instanceof Shape

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

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

o1

01

class

x 2

y 3

class

x 0

y 0

spin 3.0f

o2

class

x 0

y 0

spin 3.0f
Shapes and Points

`r instanceof Shape`

**Shape class**
- `S.move`
- `S.area`
- `S.dist`

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

```
S
```

```
Point
```

```
Shape
```

```
class
```

```
spin
```

```
x
```

```
y
```

```
x
```

```
y
```

```
s
```

```
r
```

```
class
```

```
x
```

```
y
```

```
spin
```

```
3.0f
```

```
0
```

```
0
```

```
2
```

```
3
```

```
0
```

```
0
```

```
3.0f
```
Shapes and Points

r instanceof Shape

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

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

Object

s

r

class

x

2

y

3

class

x

0

y

0

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
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);
The problem

```java
interface Trivial {
    void x();
}

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

Boring b = new R();
b.a(2, 3.14);
```
### Boring iface

<table>
<thead>
<tr>
<th>a</th>
<th>Q.Class</th>
<th>R.Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q/class</td>
<td>Object</td>
</tr>
<tr>
<td>a</td>
<td>Q.a</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>Q.b</td>
<td>R.b</td>
</tr>
</tbody>
</table>

### Trivial iface

<table>
<thead>
<tr>
<th>x</th>
<th>Q.Class</th>
<th>R.Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q/Boring</td>
<td>R/Trivial</td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

Object-Oriented Languages (Chapter 14)