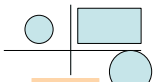
abstract classes

Every shape has a position (x, y) in the plane, so use a superclass Shape to hold the point.
Subclass has necessary fields to describe shape.

Teach using the problem of using objects to represent shapes in the plane



Rect@z
Object
fields for Shape (x, y) coords
fields for Rect length, width

Circle@y
Object
fields for Shape (x, y) coords
field for Circle radius

Circle@x
Object
fields for Shape (x, y) coords
field for Circle radius

Every subclass has an area() function

We are dealing with shapes that have areas: Circles, Rectangles, Triangles, Polyhedrons, Squares, etc.

Therefore, each subclass has a function area(), which returns its area.

Circle@x
… Object
… Shape
… Circle
area()

Rect@z
… Object
… Shape
… Rect
area()

Rect@z
… Object
… Shape
… Rect
area()

Rect@y
… Object
… Shape
… Rect
area()

Making our points with scaled-down classes

```
public class Shape { }

public class Circle extends Shape {
    public double area() {
        return 1;
    }
}

public class Rect  extends Shape {
    public double area() {
        return 1;
    }
}
```

Circle@x
… Object
… Shape
… Circle
area()

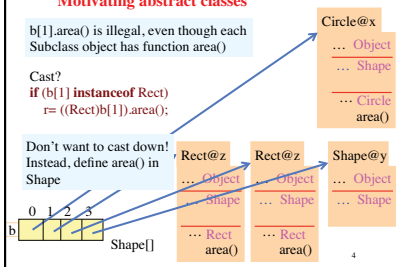
Rect@y
… Object
… Shape
… Rect
area()

Motivating abstract classes

b[1].area() is illegal, even though each Subclass object has function area()

Cast?
```
if (b[1] instanceof Rect)
   r= ((Rect)b[1]).area();
```

Don't want to cast down! Instead, define area() in Shape



4

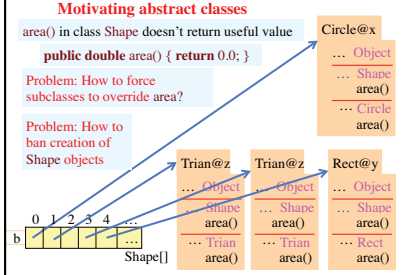
Motivating abstract classes

area() in class Shape doesn't return useful value

```
public double area() { return 0.0; }
```

Problem: How to force subclasses to override area?

Problem: How to ban creation of Shape objects



Abstract class and method solves both problems

Abstract class. Means can't create object of Shape: `new Shape(…)` syntactically illegal

```
public abstract class Shape {

    public abstract double area();
    …
}
```

Place abstract method only in abstract class.

Body is replaced by ;

Abstract method. Means it must be overridden in any subclass

6

About Interfaces

Can extend only one class

```
public class C extends C1, C2 {
    public void p() {
        …; h= m(); …
    }
}
```

if we allowed multiple inheritance, which m used?

```
public class C1 {
    public int m() {
        return 2;
    }
    …
}
```

```
public class C2 {
    public int m() {
        return 3;
    }
    …
}
```

7

Can extend only one class

```
public class C extends C1, C2 { … }
```

```
public abstract class C1 {
    public abstract int m();
    public int p() {…}
}
```

```
public abstract class C2 {
    public abstract int m();
    public int q(){…}
}
```

Use abstract classes? Seems OK, because method bodies not given!

But Java does not allow this, because abstract classes can have non-abstract methods

Instead, Java has a construct, the interface, which is like an abstract class but has more restrictions.

8

Interface declaration and use of an interface

```
public class C implements C1, C2 {
…
}
```

C must override all methods in C1 and C2

```
public interface C1 {
    int m();
    int p();
    int FF= 32;
}
```

Field declared in interface automatically public, static, final
Must have initialization
Use of public, static, final optional

```
public interface C2 {
    int m();
    int q();
}
```

Methods declared in interface are automatically public, abstract
Use of public, abstract is optional
Use ; not { … }

Eclipse: Create new interface? Create new class, change keyword **class** to **interface**

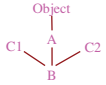
9

Casting with interfaces

class B **extends** A **implements** C1, C2 { … }
interface C1 { … }
interface C2 { … }
class A { … }

b= **new** B();
What does object b look like?

Draw b like this, showing only names of partitions:



Object b has 5 perspectives. Can cast b to any one of them at any time. Examples:

(C2) b (Object) b
(A)(C2) b (C1) (C2) b

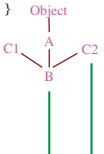
You'll see such casting later

Add C1, C2 as new dimensions:

10

Same rules apply to classes and interface

class B **extends** A **implements** C1, C2 { … }
interface C1 { … }
interface C2 { … }
class A { … }



b B@xy B
c B@xy C2

B b= **new** B();
C2 c= b;

c.m(…) calls overriding m declared in B

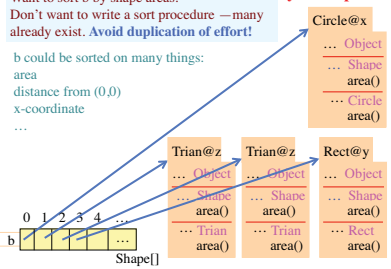
c.m(…) syntactically legal only if m declared in C2

11

Sort array of Shapes

Want to sort b by shape areas.
Don't want to write a sort procedure —many already exist. **Avoid duplication of effort!**

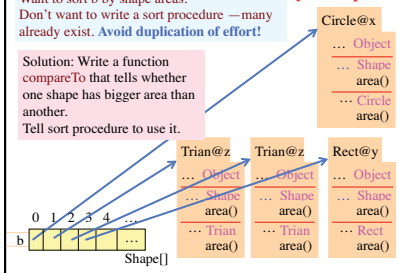
b could be sorted on many things:
area
distance from (0,0)
x-coordinate
…



Sort array of Shapes

Want to sort b by shape areas.
Don't want to write a sort procedure —many already exist. **Avoid duplication of effort!**

Solution: Write a function compareTo that tells whether one shape has bigger area than another.
Tell sort procedure to use it.



Look at: interface java.lang.Comparable

```
/** Comparable requires method compareTo */
public interface Comparable {
  /** = a negative integer if this object < c,
      = 0 if this object = c,
      = a positive integer if this object > c.
      Throw a ClassCastException if c cannot
      be cast to the class of this object. */
  int compareTo(Object c);
}
```

In class java.util.Arrays:

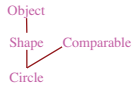
public static void sort (Comparable[] a) {…}

Classes that implement Comparable
Boolean
Byte
Double
Integer
…
String
BigDecimal
BigInteger
Calendar
Time
Timestamp
…

14

Which class should implement Comparable?

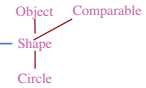
First idea: all the subclasses Circle, Rect, …



Doesn't work! Each element of b has static type Shape, and compareTo isn't available in Shape partition

Shape[] b= …
…

Use this. Shape must implement Comparable



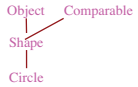
b
…
Shape[]

15

1

Shape should implement Comparable

Shape[] b= …
…
Arrays.sort(b);



In class java.util.Arrays:

public static void sort (Comparable[] a) {…}

b
…
Shape[]

16

1

Class Shape implements Comparable

```
public abstract class Shape implements Comparable {
  …
  /** If c is not a Shape, throw a CastClass exception.
      Otherwise, return -1, 0, or 1 depending on whether this
      shape has smaller area than c, same area, or greater area */
  public @Override int compareTo(Object c) {

    double diff = area() - ((Shape) c).area();
    if (diff < 0) return -1;
    if (diff > 0)  return 1;
    return 0;
  }
  …
```

Cast needed so that area() can be used. If c not a Shape, exception thrown

17

Java Library static methods:
Arrays.sort(Comparable[] a)
Collections.sort(List<Comparable> list)

Class arrays has many other useful static methods

Beauty of interfaces:

Arrays.sort sorts an array or list C[] for *any* class C, as long as C implements interface Comparable —and thus implements compareTo to say which of two elements is bigger.

18