SUMMARY: abstract classes and interfaces

Make a class abstract so instances of it cannot be created.

Make a method abstract so it must be overridden.

An interface is like an abstract class whose methods are all abstract and whose fields are all public constants. This allows multiple inheritance without ambiguity. An interface has a different syntax and a different way of using it.

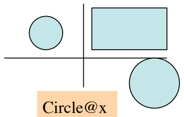
References to text and to JavaSummary.pptx

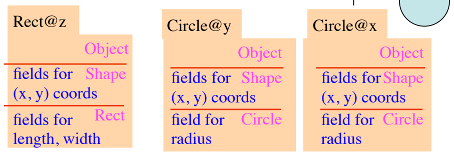
Abstract class: C.27, slides 42-44
Abstract method: C.27, slide 44
Interface declaration: D.11-D.13, D.28, slide 60
Implementing interfaces: D.14-D.15, slide 60
Casting with interfaces: none, slide 61
Interface Comparable: D.20, slide 62

abstract classes and interfaces

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 a 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

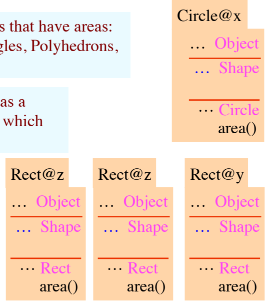
Circle@y
Object
fields for Shape
(x, y) coords
field for Circle
radius

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

Every subclass has a different area() function

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

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



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

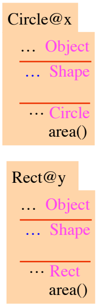
Rect@z
… Object
… Shape
… Rect
area()

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()

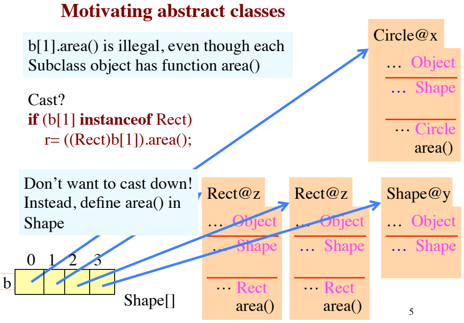
Rect@y
… Object
… Shape
… Rect
area()

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



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

Rect@z
… Object
… Shape
… Rect
area()

Rect@z
… Object
… Shape
… Rect
area()

Shape@y
… Object
… Shape

0 1 2 3
b
Shape[]

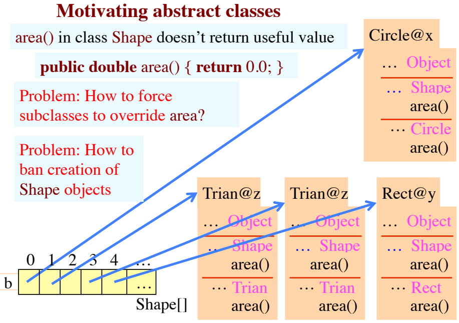
Motivating abstract classes

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



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

Trian@z
… Object
… Shape
area()
… Trian
area()

Trian@z
… Object
… Shape
area()
… Trian
area()

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

0 1 2 3 4 …
b
…
Shape[]

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

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;
   }
   …
}
```

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.

Interfaces

An interface is a fully abstract class with a slightly different syntax.

An interface can contain type signatures for methods, just like abstract methods in abstract classes, but they have to be public.

An interface can contain fields, but they have to be public, static, and final and they have to contain an initializer. So they are really just constants

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;
}
```

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

Field declared in interface automatically public, static, final

Must have initialization

Use of public, static, final optional

Methods declared in interface are automatically public, abstract

Use of public, abstract is optional

Use ; not { … }

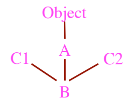
Eclipse: Create new interface? Create new class, change keyword **class** to **interface**

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:

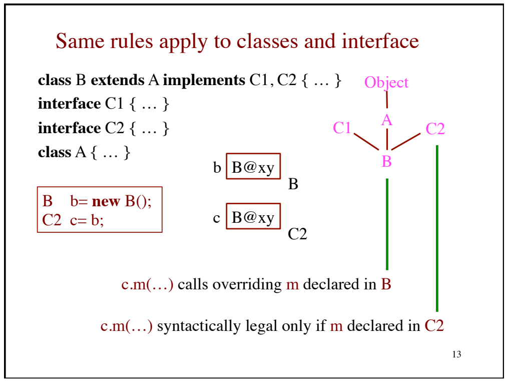


Object b has 5 perspectives. Can cast b to any one of them at any time. Examples:

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

You'll see such casting later

Add C1, C2 as new dimensions:



Same rules apply to classes and interface

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

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

b B@xy B

c B@xy C2

Object

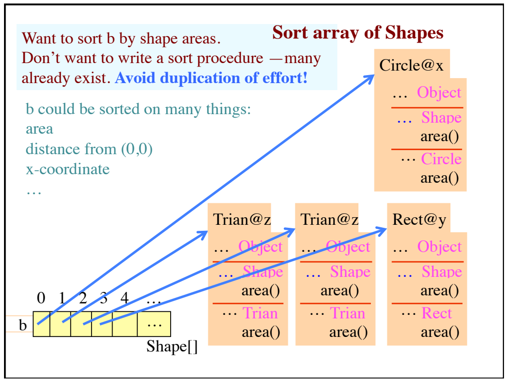
A

C1 B C2

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

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

13



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
…

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

Trian@z: … Object, … Shape, area(), … Trian, area()

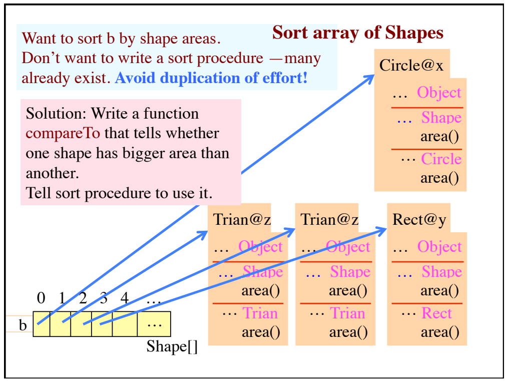
Trian@z: … Object, … Shape, area(), … Trian, area()

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

0 1 2 3 4 …

b

Shape[]



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.

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

Trian@z: … Object, … Shape, area(), … Trian, area()

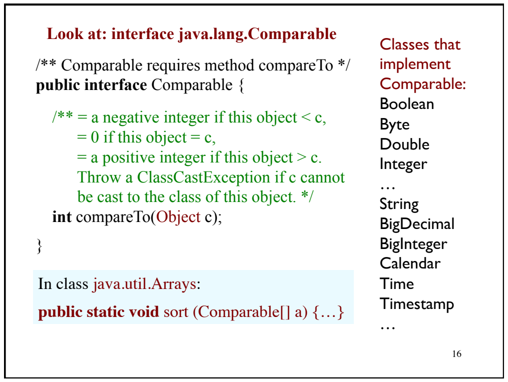
Trian@z: … Object, … Shape, area(), … Trian, area()

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

0 1 2 3 4 …

b

Shape[]



Look at: interface java.lang.Comparable

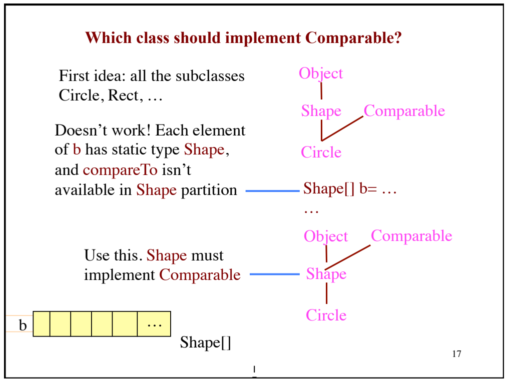
```
/** Comparable requires method compareTo */
public interface Comparable {

  /** = 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
…

16



Which class should implement Comparable?

First idea: all the subclasses Circle, Rect, …

Object

Shape Comparable

Circle

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

Object Comparable

Shape

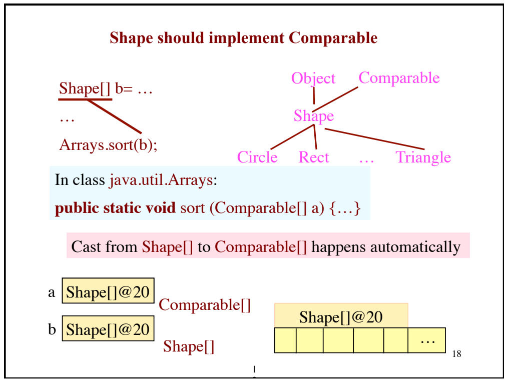
Circle

b

…

Shape[]

17



Shape should implement Comparable

Shape[] b= …
…
Arrays.sort(b);

Object Comparable

Shape

Circle Rect … Triangle

In class java.util.Arrays:

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

Cast from Shape[] to Comparable[] happens automatically

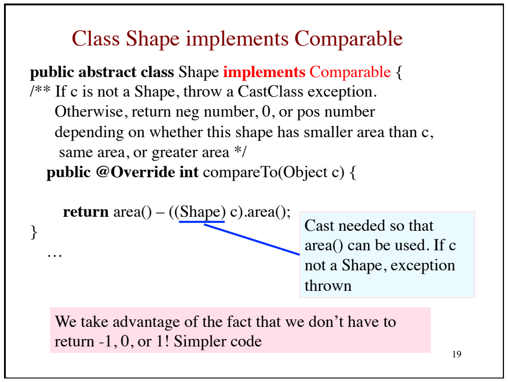
a Shape[]@20 Comparable[]

b Shape[]@20 Shape[]

Shape[]@20

…

18

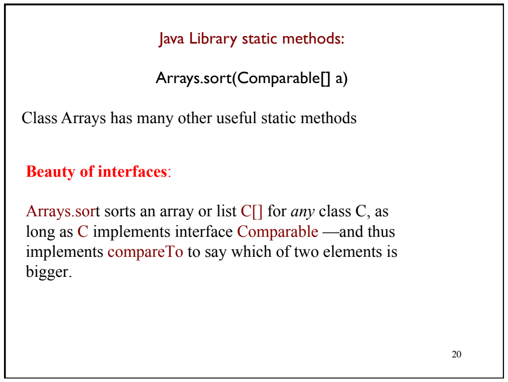


Class Shape implements Comparable

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

      return area() – ((Shape) c).area();
}
   …
```

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

We take advantage of the fact that we don’t have to return -1, 0, or 1! Simpler code



Java Library static methods:

Arrays.sort(Comparable[] a)

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.