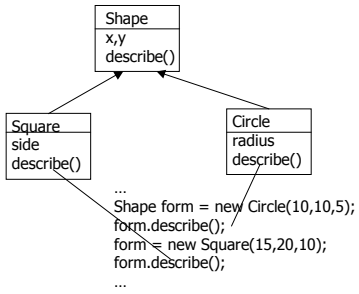
C11, Implications of Inheritance

Implications

- OO language must support polymorphic variables
- Polymorphic variables: memory requirements unknown at compile-time => allocated on the heap
- Heap allocation => reference semantics for assignment and parameter parsing
- Also: equality testing == object identity
- And: memory management necessary => garbage collection (GC)

The Polymorphic variable



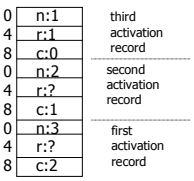
Memory Layout

- Stack-based vs. heap-based
- Stack-based tied to method entry/exit:
  - Allocate space for all local vars on entry
  - Deallocate on exit
  - Access by numeric offset (activation record)
  - Need to know space requirements at compile-time, not possible for polymorphic variables!
- Java: all objects allocated on the heap (new operator), local vars point to these objects, pointers have a fixed size
- Java has no pointers, but internally all object values are represented by pointers.

Factorial example

```
static public int factorial(int n) {
    int c = n-1;
    int r;
    if (c>0)
        r = n*factorial(c);
    else
        r = 1;
    return r;
}
```

Factorial(3)

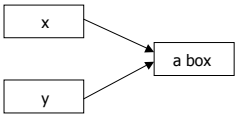


Alternative approach (e.g. C++)

- Different approaches are possible
- C++: want to use stack (efficiency)
- Assignment: extra fields are sliced off,
  - E.g. a Square is turned into a Shape,
  - Need to distinguish between virtual/non-virtual methods
  - Need destructors,
  - Pointers/references/values
  - Copy-semantics,
  - …

Assignment: reference semantics

```
Box x = new Box();
x.setValue(7);
Box y = x;
y.setValue(11);
System.out.println(x.getValue());
System.out.println(y.getValue());
…
11
11
```



Simple copy pointer value, i.e. point to same heap-location!

Clones

- If copy desired, must say so, DIY approach:

```
Box y = new Box();
y.setValue(x.getValue());
```

- If common, package into proper method:

```
public Box copy() {
    Box b = new Box();
    b.setValue(getValue());
    return b;
    }
```

Clones: Java

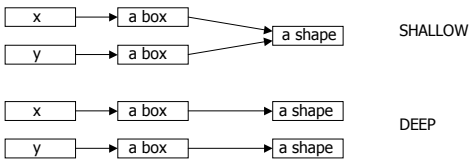
- Class Object provides protected method clone() creating a bitwise copy of the receiver, plus interface Cloneable
- Programmer must override clone() to public and implement Cloneable:

```
public class Box implements Cloneable {
    …
    public Object clone() {
        Box b = new Box();
        b.setValue(getValue());
        return b;
    }}
```

- Use: Box y = (Box) x.clone(); // must cast !!!!

Clones: caveats

- Just a shallow copy, sometimes need deep copies



Parameter passing as assignment

- Passing a variable considered similar to assignment, as same value accessible through two different names; pass value, loose some control:

```
static void sneaky (Box b) {b.setValue(11);}
…
x.setValue(7); sneaky(x);
System.out.println(x.getValue());
```

11

Equality test: object identity

- Easy for primitives:
  - 7 == (3+4)
  - 'a' == '\141'
  - 2 == 2.0
- For objects: == implements object identity, therefore:
  - new Integer(7) != new Integer(3+4)
- If we really want object equality instead of object identidy: equals method

Equality test: object equality

- Supplied by class Object, can be overridden:

```
class Circle extends Shape { …
    public boolean equals(Object arg) {
        return ((arg instanceof Circle) &&
                (radius == ((Circle) arg).radius));
    }}
```

- Careful: must be symmetric and transitive
- Heuristic: use == for numbers and null, equals in all other situations

Equality test: bug example

- Suppose:

```
class Shape { …
    public boolean equals(Object arg) {
        return ((arg instanceof Shape) &&
                (x == ((Shape) arg).x) &&
                (y == ((Shape) arg).y)) ;
    }}
```

- And no equals defined in class Square, then:

```
Square s = new Square(10,10,5);
Circle c = new Circle(10,10,5);
s.equals(c);    // succeeds
c.equals(s);    // fails
```

Garbage collection (GC)

- Heap-based memory not automatically recovered on method exit (unlike stack)
- Manual management error-prone (free in C++):
  - Forget to free: memory leaks
  - Free multiple times
  - Access freed memory
- Java compromise: have automatic garbage collection:
  - some runtime cost, but not too bad)