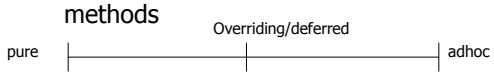
C12, Polymorphism

"many forms"
(greek: poly = many,
morphos = form)

Varieties of Polymorphism

- A spectrum of concepts, extremes:
  - Pure polymorphism: a single function can be applied to arguments of a variety of types.
  - Adhoc polymorphism: a number of different functions share the same name
- In between: overriding, deferred methods



Polymorphic variables

- Polymorphism through substitutability via polymorphic variables:
  - Declared (static, compile-time) type/class of a variable may differ from the actual (dynamic, run-time) type/class of the value it holds:

```
public void paint(Graphics g) {
   for(int I =0; i<13; i++)
        allPiles[i].display(g);
   }
```

Overloading

- One name, but two or more method bodies
- Overriding is a special case of overloading
- Different point of view: one (abstract) method specification, but depending on the argument type differing implementations (code bodies): e.g. draw() or paint()

Overloading and Coercion

- Most common example for an overloaded operator: +, can add ints, floats, doubles
- Usually associated with coercion (= automatic type conversion)
- E.g. adding ints and floats:
  - Just overloading, no coercion: 4 separate methods for int+int, int+real, real+int, real+real
  - Overloading + coercion: 2 methods for int+int, real+real, and coercion from int to real (1 => 1.0)
  - Just coercion: 1 method for real+real, + coerce int to real

Overloading from Separate Classes

- Most general form of overloading, several classes that are NOT linked by inheritance have method with same name: isEmpty() in classes Vector, HashTable, Rectangle, …
- Does not imply similarity between those classes!
- Not necessarily bad style
- Clear, short, meaningful names are good design

Parametric Overloading

- Same context (class), same name, but different numbers and types of parameters, e.g. different constructors
- Can construct rectangle with no parameters, 2 or 4 ints, a Point, a Dimension, a Point and a Dimension:

```
Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle(6,7);
Rectangle r3 = new Rectangle(10,10,6,7);
Point p1 = new Point(10,10);
Dimension d1 = new Dimension(6,7);
Rectangle r4 = new Rectangle(p1);
Rectangle r5 = new Rectangle(d1);
Rectangle r6 = new Rectangle(p1,d1);
```

Parametric overloading II

- Compiler disambiguates on types of parameters;
- Easy for constructors, but can be confusing in general (see online code example for chapter 12), basically "compile-time type matters",
- But will still call correct overridden method according to runtime type of the receiver of a method call

Overriding

- A subclass redefines a superclass method with exactly the same number and types of arguments!
- The subclass method overrides the superclass method

Replacement and Refinement

- Overriding usually replaces the superclass method, unless we call that method explicitly (called refinement):
  - super.methodName()
- Exception: constructors always have refinement semantics (even if we are not explicitly calling super(…))

Abstract methods

- Sometimes called deferred, because their implementation is deferred.
- Either inside an abstract class or interface, must be overridden later
- Useful to associate behaviors with abstract entities (draw() for Shape) when specifying classes
- Practical reason for statically typed languages: can only call draw() on polymorphic variable Shape, if draw() is defined for Shape

Pure polymorphism

- One method can be called with various types of arguments. E.g. valueOf method in class String:

```
public static String valueOf(Object o) {
    if (o == null) return null;
    return o.toString();
}
```

- Java: argument usually some high-level class (like Object above)
- variety achieved because we call another method (toString()) that is overridden for a lot of classes

Efficiency

- Programming (and design of programs) always involves compromises
- polymorphism:
  - + ease of development and use
  - + readability
  - + reuse
  - - efficiency (usually not an issue, but …)

Templates (aka Generics)

- Currently NOT in Java (but maybe in 1.5)
- C++:

```
template <class T> class box {
    public:
        box (T init) {value = init;}
        T getValue() {return value;}
    private:
        T value;
    };
```

- Use:

```
box<int> aBox(5);              // create a box with an int
box<Shape> aBox(aCircle); // create a box with a Shape
```

- Better and safer code (more efficient/less casts/errors caught at compile time), e.g. can have explicit "Vector of Shapes"

Summary

- In OO languages: polymorphic variables
- Overloading
- Overriding
- Parametric overloading
- Polymorphism: optimises development time and reliability for some cost in runtime efficiency