Subtyping

Object oriented languages permit subtyping.

```java
class Rectangle {
    private int x,y;
    int area() { ... }
}
class Square extends Rectangle {
    ...
}
Square is a subclass of Rectangle.
Since all methods on Rectangle are inherited by Square (unless explicitly overridden)
Square is a subtype of Rectangle.
```

Inheritance

```java
class Circle {
    float x, y; // center
    float r; // radius
    float area() {
        return 3.1415 * r * r;
    }
}
class ColoredCircle extends Circle {
    color c;
}
class Test {
    static main() {
        ColoredCircle t;
        ... t.area() ...
    }
```

Type Equivalence

When are two types “equal”? 

```java
type Vector = array [1..10] of real;
type Weights = array [1..10] of real;

var x, y: Vector;
z: Weight;
• Name Equivalence: When they have the same name, 
x and y have same type, but z has different type.
• Structural Equivalence: When they have the same structure,
x, y and z have same type.
```

Structural Equivalence

\( S \equiv T \) iff:

- \( S \) and \( T \) are the same \textit{basic type};
- \( S = \text{user}(\text{type}_1), T = \text{user}(\text{type}_2) \) and \( \text{type}_1 = \text{type}_2 \);
- \( S = \text{array}(S_1), T = \text{array}(T_1) \) and \( S_1 \equiv T_1 \);
- \( S = \text{pointer}(S_1), T = \text{pointer}(T_1) \) and \( S_1 \equiv T_1 \);
- \( S = \text{tuple}(S_1, S_2), T = \text{tuple}(T_1, T_2) \) and \( S_1 \equiv T_1 \) and \( S_2 \equiv T_2 \);
- \( S = \text{arrow}(S_1, S_2), T = \text{arrow}(T_1, T_2) \) and \( S_1 \equiv T_1 \) and \( S_2 \equiv T_2 \).
Resolving Overloaded Names

What entity is represented by move in r.move(3, 10)?
- Determine the type of r.
  r has to be of type user(c).
- Determine the nearest superclass of class c that has a method with name move
  such that move is a method that takes two int parameters.

Structural Subtyping

\( S \subseteq T \) iff:
- \( S \) and \( T \) are the same basic type,
- \( S = \text{user}(\text{type}_1) \), \( T = \text{user}(\text{type}_2) \) and \( \text{type}_1 \subseteq \text{type}_2 \),
- \( S = \text{array}(S_1) \), \( T = \text{array}(T_1) \), and \( S_1 \subseteq T_1 \);
- \( S = \text{pointer}(S_1) \), \( T = \text{pointer}(T_1) \), and \( S_1 \subseteq T_1 \);
- \( S = \text{tuple}(S_1, S_2) \), \( T = \text{tuple}(T_1, T_2) \), and \( S_1 \subseteq T_1 \) and \( S_2 \subseteq T_2 \);
- \( S = \text{arrow}(S_1, S_2) \), \( T = \text{arrow}(T_1, T_2) \), and \( S_1 \subseteq T_1 \) and \( T_2 \equiv S_2 \).

Resolving Names

What entity is represented by t.area()?
(assume no overloading)
- Determine the type of t.
  t has to be of type user(c).
- If c has a method of name area, we are done.
  Otherwise, if the superclass of c has a method of name area, we are done.
  Otherwise, if the superclass of superclass of c...
  \( \Rightarrow \) Determine the nearest superclass of class c that has a method with name area.

Overloading

```java
class Rectangle {
    int x, y; // top lh corner
    int l, w; // length and width

    Rectangle move() {
        x = x + 5; y = y + 5;
        return this;
    }

    Rectangle move(int dx, int dy) {
        x = x + dx; y = y + dy;
        return this;
    }
}
```
Abstract objects and Concrete Representations

Abstract classes declare methods, but do not define them.

Example:

- closedGraphical declares "area" method, but cannot define the method.
- The different "area" methods are defined when the object's representations are concrete: in rectangle, ellipse, etc.

When "area" method is applied to an object of class closedGraphical, we method to be called is the one defined in rectangle, triangle, ellipse, etc.

... which can be resolved only at runtime!

Types in OO Languages: The Whole Story

Decaf implements a small part of the type system for an OO language.

- Subtype rule: Wherever an object of type $t$ is required (as a parameter of a method, return value, or rhs of assignments), object of any subtype $s$ of $t$ can be used.

Inheritance and Overloading

What entity is represented by $f$ in $t$.f($a_1$, $a_2$, ..., $a_n$)?

- Let the type of $E$ be user($c$).
- $f$ is the method in the nearest superclass of class $c$ such that type of $f$ is a supertype of type($a_1$) $\times \cdots$ type($a_n$) $\rightarrow \bot$.

Inheritance: Another Example
Types in OO Languages: The Whole Story (contd.)

- **Method Selection rule:** If `class B` inherits from `class A` and overwrites method `m`, then for any `B` object `b`, method `m` of `B` must be used, even if `b` is used as an `A` object.

  ```java
  class A {
    int m() { ... }
  }
  class B extends A {
    int m() { ... }
  }
  class C {
    int f(B b) {
      A a;
      a = b;
      ... a.m() ...
    }
  }
  ```

Types in OO Languages: The Whole Story (contd.)

- **Dynamic Binding rule:** A method of object `obj`, which can be potentially overwritten in a subclass has to be bound **dynamically** if the compiler cannot determine the runtime type of `obj`. 