Type checking

```
v1 : t1
v2 : t2
def f(v:t2){
  |
}
```

Q.1. Can we assign: v1 = v2 ?

Q.2. ... = ... v1+v2 ... ?

Q.3 f(v1)

Answers to above questions depend on whether a PL is strongly typed or weakly typed + on the type compatibility rules of the PL

Strongly Typed PLs

- when they are strict about types.
  - i.e. we cannot do any of the above...

Weakly Typed PLs

when PLs are not strict. e.g. Python, JS

When to do type checking ?

Static Typing (statically typed PLs)

- when all type checking is done at compile time
- e.g. Java, SML, C, C++

Dynamic Typing (dynamically typed PLs)

- when type-checking is done during run-time

Type Equivalence

- defines when two types in a PL/program are equivalent.

e.g. are float + int equivalent?

```
float a;
int b;
a = b; ??
```

Two types of Type Equivalence

Name Equivalence

two type expressions are equivalent if they have the same name.

e.g. above a = b is not allowed as float + int are different type names

Structural Equivalence

Two types are structurally equivalent if they are made up of the same parts.

---

e.g. C

```
typedef float celsius;
typedef float fahr;

celsius c = 100.0;
fahr f;

f = c;  OK in C
```

since C uses structural equivalence.

Java

```
public class Celsius {
    private float temp;

} //class Celsius
public class Fahr {
    private float temp;

} //class Fahr.
```

```
Celsius c = new Celsius(100.0);
Fahr f;
f = c;
```

✗ not allowed since Java uses name equivalence.

In fact in Java, we can then write

In Celsius:

```
public float toFahr() {
}
```

In Fahr

```
public float toCelsius() {
}
```

use

```
f = c.toCelsius();
c = f.toFahr();
```

Some PLs have both!

e.g. Ada

```
type celsius = real;
type fahr  = real;
```

In Ada, we use structural equivalence for these types.
BUT, Ada also allows derived types

```
type celsius = new real;
type fahr  = new real;
```

Now celsius + fahr are new + different types even though they are structurally equivalent!

Another example w/ functions

C/Java

```
int max(int a, int b){
  |
  3
```

can we do: float x, y = . . .
max(x, y)?

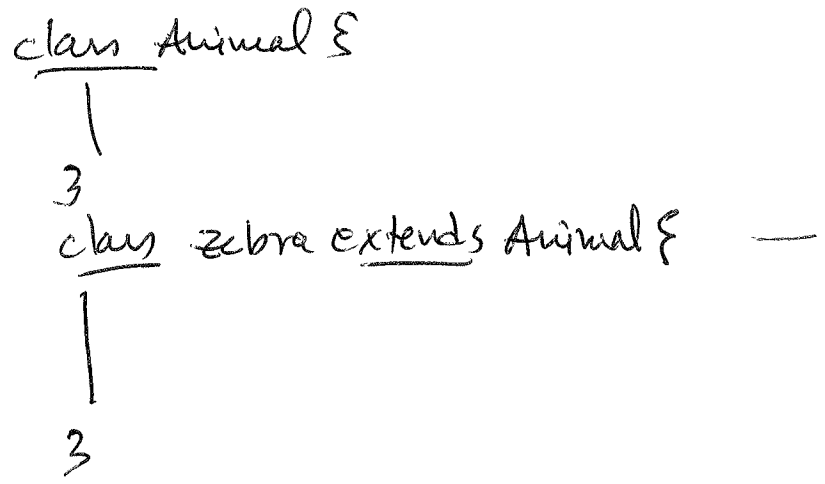
Need to define something more practical than name/structural equivalence — Type Compatibility Rules

e.g. Two types are compatible if:
1. The two types are equivalent
2. One is a subtype of the other
3. Both are arrays with the same types & elements.

Java type compatibility

1. Identical types are compatible
   i.e. int and int
   float and float

2. subtypes + supertypes



zebra is a subtype of the supertype Animal

i.e. we can do
Animal a = new zebra(); ✓
But not Zebra z = new Animal(); ✗

3. Interfaces
   any class that implements an interface is a subtype of that interface

4. Type Casting
   - explicitly convert a variable to another
   e.g.
```
float f;
int i;
i = (int) f;
```
but cannot do
```
String s;
i = (int) s;
```
etc