Types (Java)

generic:

- static checking at definition of methods + types
- code generation
- we assume every type can be represented the same way
  templates:
- static checking at compile time / type instantiation
- more efficient
- C++

```
List <String> ls = ...;
List <Object> lo = ls; <- not valid, List <String> is not a subtype of List <Object>
lo.add(new Thread());
String s = s.get();

Object -> Thread
Object -> String

we tried to convert a Thread to a String without the compiler saying a warning
```

subtypes:

- if $x \subseteq y$, then every value of type x should be useful in the context of y
- OR every operation on y should work on x values

`List <String>` $\not\subseteq$ `List<Object>`

`char *` - pointer to char
`const char *` - pointer to char that can't be modified via itself

ex:

```
char c = 'a';
const char *p = &c;
print(*p);
```

```
f(&c);
print(*p) <- if f modifies the address of c the output changes
```

`char *` $\subseteq$ `const char *`

- `const char *` is on the right side because it has fewer operations

```
List <String> ls = ...;
void printList(List <Object> l) {
    for (Object o : l)
        System.out.println(o);
}
```

we can't call `printList(ls)` because we don't now if it will modify `ls` in unknown ways. So it only works then the input is `List <Object>` explicitly.

fixing the function:

```
void printList(List <?> l) {...}
```

```
public void printShapes(Collection <?> shapes) {
    for (Shape s : shapes)
        printShape; <- won't work
}
```

Bounded Wildcard

code fix:

```
Collection <? extends Shape> shapes)
```

code that won't work:

```
void cvt([]arr, Collection <?> n) {
    for (Object o : arr)
        co.add(o);
}
```

code fix:

```
<T> void cvt(T []arr, Collection <T> n) {
    for (T o : arr)
        co.add(o);
}
```

```
Square []as;
Collection <Shape> cs = ...;
cvt(as, cs) <- won't work
```

code fix:

```
<T> void cvt(T []arr, Collection <? super T> cs) {...}
```

Implementation Notes:

```
-----------------                     --------------
| type  |       | --> type descriptor |             |
-----------------                     --------------
```

- there is only 1 type descriptor for `List <>` to keep it simple
- type erasure at runtime
- this complexity is confusing, can we do something similar?

Duck Typing

- the type of an object is a bad notion
- lets just have objects _ methods
  - o.quack
  - o.waddle
  - o.hasRoundBeak()

El Capitan

- world's fastest computer
- 43808 boxes containing AMD M!300 APC each with 24 AMD Zen 4 x86-64 cores
- 228 GPU
- SIMD (single instruction multiple data)
- 4 matrix cores

- 64 stream processors
- 11039616 cores total
- 30 MW
- 1.75 Exa Flops Linpack
  - Exa - 10^{18}
  - Flops - floating point operations (double)
- 45 Giga Flops / W

Jedi (EuroHPC)

- 19514 cores
- 4.5 Peta Flops
- 7.5 Giga Flops / W

Sun Microsystems (Oracle)

- workstations
- servers (multicore/CPU)
- SPARC~x86-64
- Solaris~Linux
- "The network is the computer"
- looked to the future, wanted their software to work on toasters
  - problems:
    - embedded world wants cheap CPUs
    - problems with this:
      1. multiple architectures - compile program N times, distribute N copies
      2. 20 KB/s Internet + bug executables (bad)
      3. C/C++ takes too long to build/test
- to try to find a solution they went to Silicon Valley to see what the competition was doing

Xerox PARC

- creations:
  - network - Ethernet + mouse + bitmapped display + Smalltalk (IDE)

Smalltalk

- object oriented
- interpreted bytecode -> run on interpreter(ASM)

- garbage collector
- subscript checking

OAK

- C++ syntax + Smalltalk + more
- renamed to Java
- to show its practicality it was used to make a browser, Hot Java

Hot Java

- browser built in Java
- died
- Java - flexible outlets
- more reliable
- showed people that Java is useful, more so its multithreading applications

Mosaic

- popular web browser at the time
- wrote in C++
- crashed frequently
- ancestor to Mozilla / Internet Explorer

Simple Java

- not C/C++
    - single inheritance
    - no pointers -> references
    - garbage collector
    - primitive types are portable

```
byte (8) short (16) int (32) long (64) float (32) double (64) bool (1)
```

- sizes will be simulated to match the machine