Multiple Inheritance

1. Object Representation
2. Ambiguity

0 Object Rep.

SI c: Serializable
but SI rep doesn't match
Serializable.

⇒ one vptr per superclass.
2 Name Clashes
  - Disallow.
  - Implicit Ordering (Clos, Scala traits)
    - ordered decls.
    - brittle.
  - Explicit (C++)
    - programmer resolves ambiguity
  - class SerialImage {
    virtual print() {
      Image::print();
    }
  }
3. Diamond Pattern

- possible layouts:

Neither gives us:
- one x φ y
- valid substitutivity.
Virtual Base Classes.

Pt

CPoint

Blinking Point

BCPoint

\[
\begin{array}{c}
x \\
\hline
y
\end{array}
\]

\[
\begin{array}{c}
\text{Color} \\
x \\
y
\end{array}
\]

\[
\begin{array}{c}
\text{freq} \\
x \\
y
\end{array}
\]
Java


-Oak.
Design Criteria

- Simple.
- Portable.
- Safe / Reliable.
- Efficient?
- Complete.
Java OO Features

1) Dynamic lookup for all methods.
2) Single Inheritance
3) Encapsulation: public/protected/private
4) Subtyping - via inheritance via interfaces.

Details

1) Constructor must call superclass constructor at its start.
   - init private data in super
   - establish invariants.

2) Object class is superclass of all classes.
3. Final Methods.
   - can't be overridden.
   - guarantee behavior in subclasses.

4. Final Classes.
Inheritance & Subtyping

- Two uses for MI:
  1. Code reuse.
  2. Multiple supertypes.
- Interfaces gives you #2
  - "protocols"
  - no rep info.

```java
List<MouseListener> list = ...;
list.add(new TextEditor);
for (MouseListener m : list) {
    m.buttonClicked(e);
}
```
- Simpler design, language
- Slower dispatch
- May need to duplicate code more.