The Power of Polymorphism

Food f;

\[ \text{can hold a reference to anything that's an instance of Food} \]

\[ \Rightarrow \]

Food

\[ \Rightarrow \]

Meat, Fruit

Object

"is-a"

Food

getCalories()

"is-a"

Meat, Fruit

Object x;

\[ \text{can hold any object} \]
Polymorphism

many shape

A variable can hold refs. to objects of many "shapes" or types.

Food f = new Meat(\ldots);

OK because of inheritance...

f. getCalories()

Power comes from overriding methods,
have a `List<Food> l`

for (Food f : l)
    Sum += f.getCalories();

- **Generality** — write programs in terms of general types

- **Extensibility** — create new subclasses with different behavior (by overriding methods) without modifying the general program.
Subclasses can...
  - augment the available functionality
  - override functionality (specialize)
  - be abstract — not instantiable... can’t say new Shape(...)
for (Graphic g : list_of_Graphics) {
    g.paint(...);
}
Technical issues regarding class hierarchies + polymorphism

Principles:

1. Type-checking is done at compile time (statically — by looking at the program text)

2. Method dispatching happens at run-time based on the actual type of the object (not the type of the variable)
Super — refers to the parent class

1. Use in a constructor

```java
public MyClass extends YourClass { 
  public MyClass (int x, int y) { 
    super (x + y); // Calls constructor from YourClass
    // Can't refer to "this"
  }
}
```

2. Use in a method to call inherited method we have overridden

```java
public reset () { 
  super.reset();
}
```
\[ \text{instanceof respects class hierarchy} \]

\[
\begin{align*}
f &= \text{new Foo()}
\text{b} &= \text{new Bar()}
\text{b2} &= \text{new Baz()}
\end{align*}
\]

\[
\begin{align*}
(f \text{ instanceof Foo}) & \implies \text{true} \\
(b \text{ instanceof Foo}) & \implies \text{true} \\
(b2 \text{ instanceof Bar}) & \implies \text{false} \\
(\text{null instanceof Foo}) & \implies ??
\end{align*}
\]