1. Polymorphism
   1a. Generics
2. Abstract Classes
3. Interfaces *
4. Marker Interfaces, Shallow vs Deep Copying
Polymorphism —

parent class

child classes
Program needs a list of shapes

```pseudo
def draw():
    Vector shapes = new Vector()
    for shape in shapes:
        shape.draw()
```
boolean add(Object o)

Vector shapes = new Vector();
Circle c = new Circle(10);
shapes.add(c);
String s = "hello";
shapes.add(s);
Generics

Vector<Shape> shapes = new Vector<Shape>();

(templated classes)

allow automatic type safety
\texttt{List<\textit{Equipment}> r1 = new List<\textit{Equipment}>();}
Abstract Classes

Class Rhombus extends Shape

}  What does the
draw method do?

What does area method
calcul r() do?
There are no good implementations for
Shape.area()
Shape.draw()
So, why define those two methods in Shape class?
We need them to be able to
test shapes polymorphically
We could implement the functions as "do-nothing" methods

```java
public void draw()
```

Solution: abstract class
public abstract class Shape {
    public Color getColor() {
        return my-color;
    }
    public void abstract draw() {
        // This forces all sub classes to implement draw or they won't compile.
    }
}
Can't instantiate abstract classes

They can have constructors
which are inherited by child classes

Abstract methods can't have code & must be overridden in child classes

Abstract classes may have concrete methods
Interfaces

Equipment

Insured Object

how do I create a

Insured Equipment

Grant Object
Interfaces are a technique to bypass limitations of single inheritance.