When a method is performed on an object, the resultant behaviour is defined by the implementation of that method in the object's class.

**Standard Method Invocation Via Single-Dispatch**
The Limitation

Sometimes the behavior must also be determined by the type of the parameter object.
GetTarget() and GetParameter() can return any of the subclasses.

```
public void commonMethod(q)
{
    SuperClass q = GetParameter();
    SuperClass a = GetTarget();
}
```

Suppose that we have a piece of code that reads:

- Suppose that there are subclass Superb, Suba and Subc.
- Suppose that there is a class hierarchy with Superclass Superclass and

The Situation
{ ... } void commonMethod(SubcParam)
{ ... } void commonMethod(SubbParam)
{ ... } void commonMethod(SubaParam)

The sibling class as input:

But there are several versions of commonMethod(), each taking one of

Each of the subclasses has its own implementation of commonMethod().

The Situation Continues...
- etc.
  
  will be called;
- if a reference an object of type SubC, then SubC's commonMethod()
  
  will be called;
- if a reference an object of type SubA, then SubA's commonMethod()

  the type of the object referenced by a.

  We know that the commonMethod() that executes will be determined by

  The Situation Continues...
• a common method (p);

  . . .

  void method(Superclass a, Superclass p)
   . . .

   }

   (p)

   – For example,

   Implementation.

obscure referenced by p. So, it cannot choose the appropriate method

But, the Java compiler cannot necessarily determine the type of the

The Problem


A Strawman Solution

and to execute different code as a result:

One solution would be to declare a single commonMethod() with


•
issue).
also have to be modified to accommodate the new class (a maintenance
— If a new class of parameter object was added, then the test code would

This is, however, contrary to the principles of object-oriented

Don't do that!
A Better Solution: Double-Dispatching
For example, for example:

```java
class Subclass extends Superclass {
    void commonMethodfromSuper(SuperParam param) {
        param.commonMethodfromSuper(this);
    }
}
```