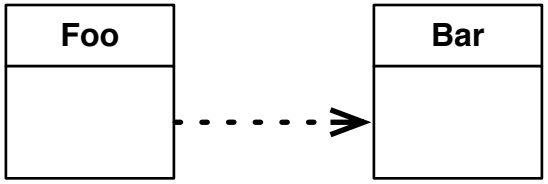
The Many Arrows of UML

UML Diagrams can also be used to the show relationships between classes. There are four important relationships that we will use in CS302:

Dependence

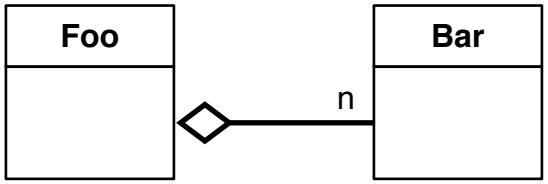


Dependence is the most basic relationship between classes. Class `Foo` depends on class `Bar` if it uses instances of `Bar` at some point in its code. If the public interface for `Bar` ever changes, the code in `Foo` may need to be updated.

Keep in mind that this does not imply that `Bar` depends on `Foo`; `Bar` probably does not know what classes are using it.

```
public class Foo {
  public void doStuff() {
    Bar b = new Bar(...);
    ....
  }
}
```

Aggregation

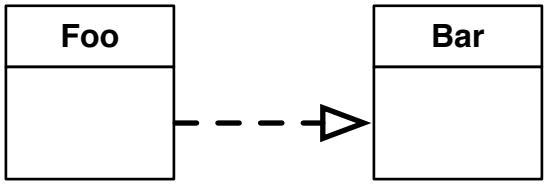


Aggregation shows that an instance of `Foo` owns instances of `Bar`. Aggregation requires that the `Foo` instances have a member variable that will keep a reference to a `Bar` object between method calls. The *Multiplicity Indicator* **n** is used to show how many instances a `Foo` object owns:

0..1	One instance, may be a null reference
1	One instance, cannot be null
4	(or another number): A fixed number of instances
*	Multiple instances, must be in an array

```
public class Foo {
  private Bar b; // n = 0..1

  private Bar x; // never null: n = 1

  private Bar b1, b2, b3;  // n = 3

  private Bar[] arr; // n = 0..*

}
```

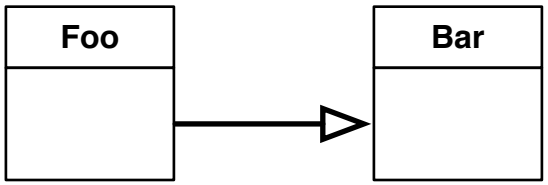
Implementation



UML can also be used to identify interface *Implementation*. If Class `Foo` implements Interface `Bar`, every method listed in `Bar` must have actual code in `Foo`'s definition. If the methods listed in interface `Bar` ever change, `Foo` will need to be changed as well. A single class can implement multiple interfaces, and can also have its own methods that do not appear in any interface. The methods in `Foo` must have the exact same signature as those in `Bar` for the compiler to find them

```
public interface Bar {
  String performQuery();
}

public class Foo implements Bar {
  // required by interface Bar
  public String performQuery() {
    return "....";
  }
}
```

Inheritance



Inheritance shows that Class `Foo` is a child class of `Bar`. `Foo` will gain all the methods in `Bar`, and a object of type `Foo` can be used anywhere a `Bar` object can be used. `Foo` may also add new methods, and can overrride the methods in `Bar` with its own code.

```
public class Bar {
  public String performQuery() { ... }
  public void doStuff() { ... }
}

public class Foo extends Bar {
  // overrides code in Bar
  public String performQuery() { ... }

  // code for doStuff() is inherited
}
```