We *cannot* do the same in Java, as the following will not compile:

```
public class Communicate  {
    public <T> void speak(T speaker) {
        speaker.talk();
    }
}
```

Confusingly, this *does* compile, as under the hood, generic types are converted to instances of `Object` in Java (this is called erasure):

```
public class Communicate  {
    public <T> void speak(T speaker) {
      speaker.toString(); // Object methods work!
    }
}
```

So, we have to do something like this:

```
interface Speaks { void speak(); }

public class Communicate  {
    public <T extends Speaks> void speak(T speaker) {
      speaker.speak();
    }
}
```

But this pretty much defeats the advantages of generics, as we are defining a type via the `Speak` interface. The lack of generality also shows in that Java primitive types cannot be used with the generics mechanism. As a workaround, Java offers wrapper classes, true object classes that correspond to primitive types. Converting between primitives and wrappers used to be a chore in Java programming. In recent versions of the language it is less so, thanks to the auto-boxing feature that performs automatic conversions in certain circumstances. Be that as it may, we can write `List<Integer>`, but not `List<int>`.

Latent typing has been popularized recently thanks to its widespread adoption in the Ruby programming language. The term "duck typing" is a tongue-in-cheek reference to inductive reasoning, attributed to James Whitcomb Riley, which goes:

> If it walks like a duck and quacks like a duck, I would call it a duck.

To see the importance of duck typing, take an essential feature of object-oriented programming, *polymorphism*. Polymorphism stands for the use of different types in the same context. One way to achieve polymorphism is through inheritance. A subclass can be used (more precisely, should be used, because programmers can be careless) wherever a superclass can be used. Duck typing offers an additional way to achieve polymorphism: a type can be used anywhere it offers methods fitting the context. In the pet and robot example shown earlier in Python and C++, `Dog` and `Robot` do *not* share a superclass.

Of course it is possible to program your way around duck typing only the inheritance type of polymorphism. A programmer, however, is wealthier if she has more tools at her disposal for