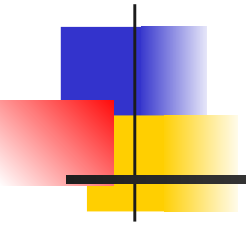


Using Inheritance

Liskov Substitution Principle

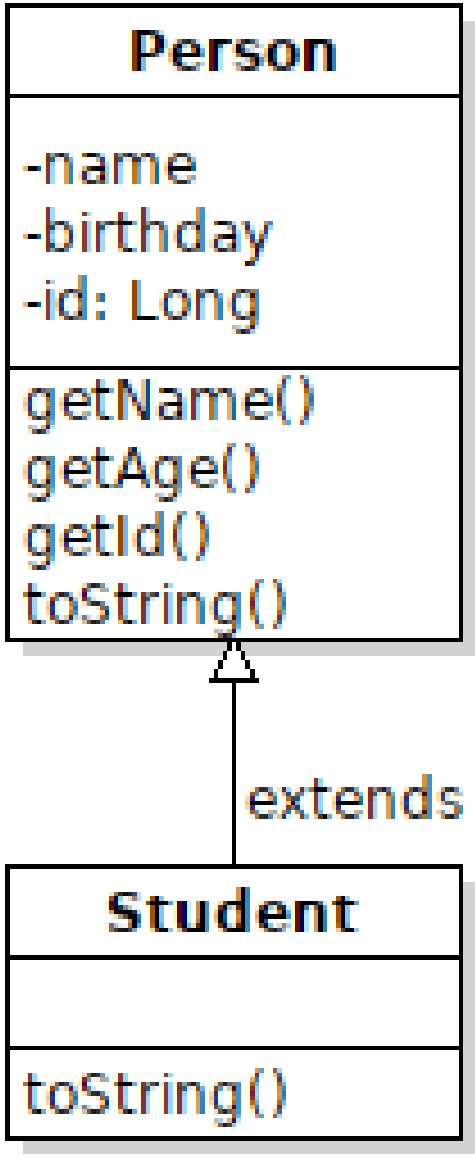
If an application uses an object of type T and S is a subtype (i.e. subclass) of T,

Then the application should work correctly given an instance of S in place of T.

greet(Person pee) { /* greet a Person */ }

Student s = new Student("Joe",59101234");

greet(s); // <------ this should work



Polymorphism

We can invoke a method of an object (reference) without knowing the actual class that will perform the method call.

At run-time, the application invokes the method of the actual object type.

Simple Example

Every class is automatically a subclass of Object.

So we can write:

```
Object obj = LocalDate.now( );
// obj refers to a LocalDate object
System.out.println( obj.toString() );
```

Which class's `toString()` method will be invoked?

It is the toString() of the actual class that obj refers to, namely **`LocalDate`**.

This is polymorphism -- we can invoke obj.toString() without knowing (or caring) which class will perform the action.

Inheritance enables Polymorphism

```
public void greet(Person person) {
    System.out.println( person.toString() );  // #1
    System.out.printf("Hello, %s\n",
                          person.getName() );  // #2
```

If we do this:

```
Student s = new Student("Nisit");
greet( s );
```

Then:

#1 will call toString() of Student -- Student has its own toString method.

#2 will call getName() of Person, because Student inherits it.

Don't Repeat Yourself (DRY)

Don't write duplicate logic or duplicate code.

A good way to eliminate duplicate code is:

1. extract duplicate code to a separate method
2. if some small part of the code is not exactly the same, use polymorphism so that the "differing" part looks the same in the context of the method.

Don't Repeat Yourself (DRY) Example

Don't write duplicate logic or duplicate code:

```
Task1 task1 = new Task1(10000000);
System.out.println(task1);
stopwatch.start();
task1.run();
stopwatch.stop();
System.out.printf("Elapsed time %.6f sec\n\n",
                        stopwatch.getElapsed());
```

Not DRY

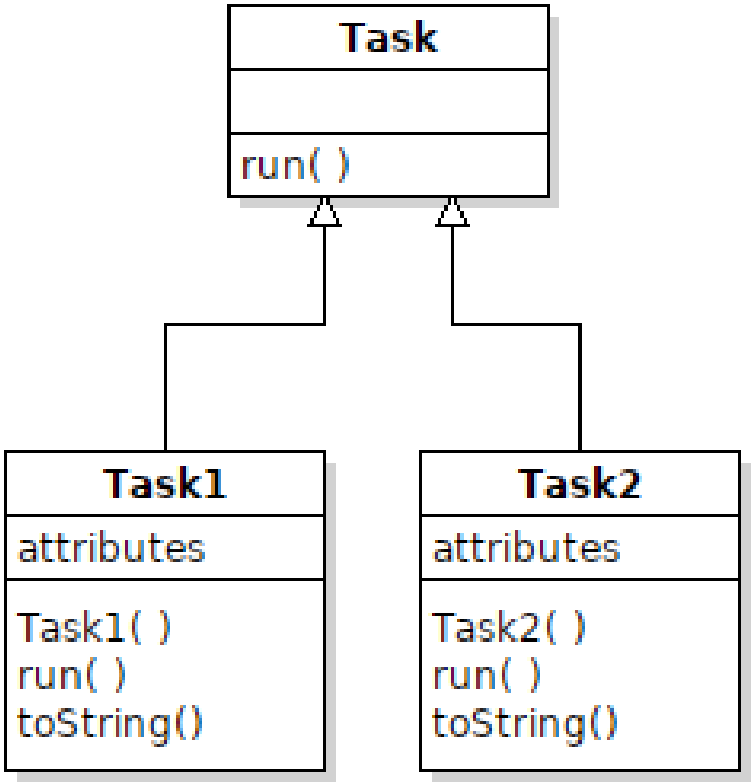
Look familiar?

```
Task task2 = new Task2(10000000);
System.out.println(task2);
stopwatch.start();
task2.run();
stopwatch.stop();
System.out.printf("Elapsed time %.6f sec\n\n",
                        stopwatch.getElapsed());
```

How to Apply Polymorphism?

We need to make all the tasks "look" alike to Java, so that we can call task.run() on any kind of task, using the same code.

Define a Task superclass:



Put common code into a method

```
public void timeAndPrint( Task task ) {
   System.out.println(task);
   stopwatch.start();
   task.run();   // <-- this part is different for each task
   stopwatch.stop();
   System.out.printf( ... ); // print the elapsed time
}
// main:
Task1 task1 = new Task1(10000000);
timeAndPrint( task1 );
```