# Object Oriented Design: Listings

Bartosz Telenczuk, Niko Wilbert

Advanced Scientific Programming in Python

Autumn School 2010, Trento

# Listing 1

```
import random

class Die(object):  # derive from object for new style classes
    """Simulate a generic die."""

    def __init__(self, sides=6):
        """Initialize and roll the die.

        sides -- Number of faces, with values starting at one (default is 6).
        """
        self._sides = sides  # leading underscore signals private
        self._value = None  # value from last roll
        self.roll()

    def roll(self):
        """Roll the die and return the result."""
        self._value = 1 + random.randrange(self._sides)
        return self._value

    def __str__(self):
        """Return string with a nice description of the die state."""
        return "Die with %d sides, current value is %d." % (self._sides, self._value)

class WinnerDie(Die):
   """Special die class that is more likely to return a 1."""

   def roll(self):
        """Roll the die and return the result."""
        super(WinnerDie, self).roll()  # use super instead of Die.roll(self)
        if self._value == 1:
            return self._value
        else:
            return super(WinnerDie, self).roll()
```

# Listing 2

```
class MyIterable(object):
    """Example iterable that wraps a sequence."""

    def __init__(self, items):
        """Store the provided sequence of items."""
        self.items = items

    def __iter__(self):
        return MyIterator(self)

class MyIterator(object):
    """Example iterator that is used by MyIterable."""

    def __init__(self, my_iterable):
        """Initialize the iterator.

        my_iterable -- Instance of MyIterable.
        """
        self._my_iterable = my_iterable
        self._position = 0

    def next(self):
        if self._position < len(self._my_iterable.items):
            value = self._my_iterable.items[self._position]
            self._position += 1
            return value
        else:
            raise StopIteration()

    # in Python iterators also support iter by returning self
    def __iter__(self):
        return self
```

# Listing 3

```
class Beverage(object):

    # imagine some attributes like temperature, amount left,...

    def get_description(self):
        return "beverage"

    def get_cost(self):
        return 0.00

class Coffee(Beverage):

    def get_description(self):
        return "normal coffee"

    def get_cost(self):
        return 3.00

class Tee(Beverage):
    def get_description(self):
        return "tee"
    def get_cost(self):
        return 2.50

class CoffeeWithMilk(Coffee):

    def get_description(self):
        return super(CoffeeWithMilk, self).get_description() + ", with  milk"

    def get_cost(self):
        return super(CoffeeWithMilk, self).get_cost() + 0.30

class CoffeeWithMilkAndSugar(CoffeeWithMilk):

    # And so on, what a mess!
```

# Listing 4

```
class Beverage(object):

    def get_description(self):
        return "beverage"

    def get_cost(self):
        return 0.00

class Coffee(Beverage):
    #[...]

class BeverageDecorator(Beverage):

    def __init__(self, beverage):
        super(BeverageDecorator, self).__init__()  # not really needed here
        self.beverage = beverage

class Milk(BeverageDecorator):

    def get_description(self):
        #[...]

    def get_cost(self):
        #[...]

coffee_with_milk = Milk(Coffee())
```

# Listing 5

```
class Duck(object):

    def __init__(self):
        # for simplicity this example class is stateless

    def quack(self):
        print "Quack!"

    def display(self):
        print "Boring looking duck."

    def take_off(self):
        print "I'm running fast, flapping with my wings."

    def fly_to(self, destination):
        print "Now flying to %s." % destination

    def land(self):
        print "Slowing down, extending legs, touch down."


class RedheadDuck(Duck):

    def display(self):
        print "Duck with a read head."


class RubberDuck(Duck):

    def quack(self):
        print "Squeak!"

    def display(self):
        print "Small yellow rubber duck."
```

# Listing 6

```
class FlyingBehavior(object):
    """Default flying behavior."""

    def take_off(self):
        print "I'm running fast, flapping with my wings."

    def fly_to(self, destination):
        print "Now flying to %s." % destination

    def land(self):
        print "Slowing down, extending legs, touch down."


class Duck(object):

    def __init__(self):
        self.flying_behavior = FlyingBehavior()

    def quack(self):
        print "Quack!"

    def display(self):
        print "Boring looking duck."

    def take_off(self):
        self.flying_behavior.take_off()

    def fly_to(self, destination):
        self.flying_behavior.fly_to(destination)

    def land(self):
        self.flying_behavior.land()
```

# Listing 7

```
class NonFlyingBehavior(FlyingBehavior):
    """FlyingBehavior for ducks that are unable to fly."""

    def take_off(self):
        print "It's not working :-("

    def fly_to(self, destination):
        raise Exception("I'm not flying anywhere.")

    def land(self):
        print "That won't be necessary."

class RubberDuck(Duck):

    def __init__(self):
        self.flying_behavior = NonFlyingBehavior()

    def quack(self):
        print "Squeak!"

    def display(self):
        print "Small yellow rubber duck."

class DecoyDuck(Duck):

    def __init__(self):
        self.flying_behavior = NonFlyingBehavior()

    def quack(self):
        print ""

    def display(self):
        print "Looks almost like a real duck."
```

# Listing 8

```
class Turkey(object):

    def fly_to(self):
        print "I believe I can fly..."

    def gobble(self, n):
        print "gobble " * n

class TurkeyAdapter(object):

    def __init__(self, turkey):
        self.turkey = turkey
        self.fly_to = turkey.fly_to  #delegate to native Turkey method
        self.gobble_count = 3

    def quack(self):                          #adapt gobble to quack
        self.turkey.gobble(self.gobble_count)

>>> turkey = Turkey()
>>> turkeyduck = TurkeyAdapter(turkey)
>>> turkeyduck.fly_to()
I believe I can fly...
>>> turkeyduck.quack()
```