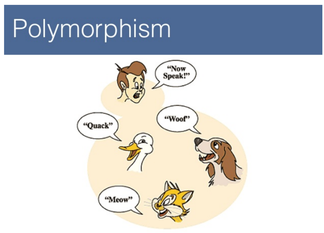
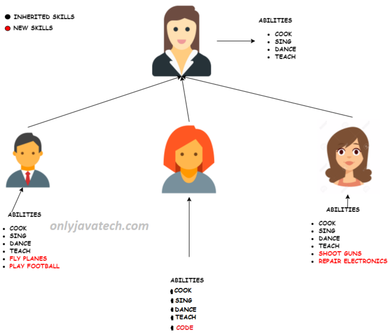
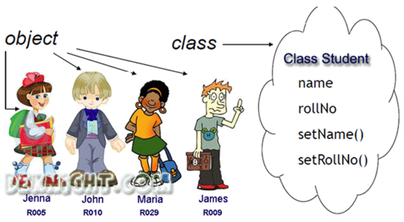
Polymorphisme



Polymorphisme merupakan tipe inheritance bilamana kita memanggil sebuah fungsi, namun dengan metode yang berbeda. Menggunakan warisan anda dapat mewarisi semua bidang data akses dan metode, ditambah anda dapat menambahkan metode dan bidang anda sendiri, sehingga warisan memberikan cara untuk mengatur kode, daripada menulis ulang dari awal.



Polimorfisme didasarkan pada kata-kata Yunani. Poli (banyak) dan morfisme (bentuk). Terkadang suatu objek datang dalam banyak jenis atau bentuk. Jika kita memiliki tombol, ada banyak output yang berbeda (tombol bulat, tombol periksa, tombol kotak, tombol dengan gambar) tetapi mereka berbagi logika yang sama.



In [1]:
```
class Shark():
    def swim(self):
        print("The shark is swimming.")
    def swim_backwards(self):
        print("The shark cannot swim backwards, but can sink backwards.")
    def skeleton(self):
        print("The shark's skeleton is made of cartilage.")
#membuat class baru dengan metode yang sama namun ouput yang diberikan berbeda
class Clownfish():
    def swim(self):
        print("The clownfish is swimming.")
    def swim_backwards(self):
        print("The clownfish can swim backwards.")
    def skeleton(self):
        print("The clownfish's skeleton is made of bone.")

sammy = Shark()
sammy.skeleton()

casey = Clownfish()
casey.skeleton()
```

```
The shark's skeleton is made of cartilage.
The clownfish's skeleton is made of bone.
```

Contoh polymorphisme dengan fungsi

In [2]:
```
class Shark():
        def swim(self):
                print("The shark is swimming.")
        def swim_backwards(self):
                print("The shark cannot swim backwards, but can sink backwards.")
        def skeleton(self):
                print("The shark's skeleton is made of cartilage.")

class Clownfish():
        def swim(self):
                print("The clownfish is swimming.")
        def swim_backwards(self):
                print("The clownfish can swim backwards.")
        def skeleton(self):
                print("The clownfish's skeleton is made of bone.")
#sebuah fungsi yang membuat semua objek memanggil metode swim
def in_the_pacific(fish):
    fish.swim()

sammy = Shark()
casey = Clownfish()

in_the_pacific(sammy)
in_the_pacific(casey)
```

```
The shark is swimming.
The clownfish is swimming.
```

Contoh lain membuat 2 objek dengan memanggil metode yang sama,namun dengan output yang berbeda.

In [3]:
```
class Bear():
        def sound(self):
                print("Groarrr")

class Dog():
        def sound(self):
                print("Woof woof!")

def makeSound(animal):
    animal.sound()
bearObj = Bear()
dogObj = Dog()
makeSound(bearObj)
makeSound(dogObj)
```

```
Groarrr
Woof woof!
```

In [4]:
```
class Shark():
        def swim(self):
                print("The shark is swimming.")
        def swim_backwards(self):
                print("The shark cannot swim backwards, but can sink backwards.")
        def skeleton(self):
                print("The shark's skeleton is made of cartilage.")

class Clownfish():
        def swim(self):
                print("The clownfish is swimming.")
        def swim_backwards(self):
                print("The clownfish can swim backwards.")
        def skeleton(self):
                print("The clownfish's skeleton is made of bone.")
sammy = Shark()
casey = Clownfish()

for fish in (sammy, casey):
    fish.swim()
    fish.swim_backwards()
    fish.skeleton()
```

```
The shark is swimming.
The shark cannot swim backwards, but can sink backwards.
The shark's skeleton is made of cartilage.
The clownfish is swimming.
The clownfish can swim backwards.
The clownfish's skeleton is made of bone.
```

dua objek, sammy dari kelas Shark, dan casey dari kelas Clownfish. Perulangan for mengulangi objek-objek ini, memanggil swim (), swim_backwards (),dan metode skeleton (). Metode yang terkait dengan kelas Shark terlebih dahulu, kemudian kelas Clownfish.

In [5]:
```
class Document:
    def __init__(self, name):
        self.name = name

    def show(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Pdf(Document):
    def show(self):
        return 'Show pdf contents!'

class Word(Document):
    def show(self):
        return 'Show word contents!'

documents = [Pdf('Document1'),
             Pdf('Document2'),
             Word('Document3')]

for document in documents:
    print(document.name + ': ' + document.show())
```

```
Document1: Show pdf contents!
Document2: Show pdf contents!
Document3: Show word contents!
```

kelas abstrak yang disebut Document. Kelas ini tidak memiliki implementasi tetapi mendefinisikan struktur (dalam bentuk fungsi) yang harus dimiliki semua bentuk. Dan Document class terdapat metode abstrak yang nantinya diperjelas dari subclass. Objek documents merupakan sebuah 3 list dari pdf dan word class.

Contoh lain dari abstrak class dan perulangan for

In [6]:
```
class Car:
    def __init__(self, name):
        self.name = name

    def drive(self):
        raise NotImplementedError("Subclass must implement abstract method")

    def stop(self):
        raise NotImplementedError("Subclass must implement abstract method")

class Sportscar(Car):
    def drive(self):
        return 'Sportscar driving!'

    def stop(self):
        return 'Sportscar braking!'

class Truck(Car):
    def drive(self):
        return 'Truck driving slowly because heavily loaded.'

    def stop(self):
        return 'Truck braking!'


cars = [Truck('Bananatruck'),
        Truck('Orangetruck'),
        Sportscar('Z3')]

for car in cars:
    print(car.name + ': ' + car.drive())
```

```
Bananatruck: Truck driving slowly because heavily loaded.
Orangetruck: Truck driving slowly because heavily loaded.
Z3: Sportscar driving!
```