Comparison JAVA & Python

Himanish Pathak/HP1235

TABLE OF CONTENTS

Comparisons _____ 3
Class _____ 4
Multiple Inheritance _____ 5
Multiple inheritance resolution _____ 6
Type() _____ 7
isinstance() _____ 8
Class methods _____ 9
Introspection _____ 10
Object and Class Encapsulation _____ 11

GENERAL COMPARISON

Dynamic Typing:

Java is a statically typed and Python is a dynamically typed. Python is strongly but dynamically typed. This means names in code are checked for type at runtime. This makes Python very easy to write and read, but difficult to analyze. The downside is that not having type information means it can be hard to tell what is going on at any given place in the code, particularly when names are ambiguous, which is frequently true. Java, in contrast, is statically typed and names in Java are bound to types at compile time via explicit type declaration. This means many type errors that would result in a runtime error–and often result in a program crash–in Python get caught during compile time in Java.

Interpreted/Compiled

Python is an interpreted language while Java is a compiler-based language.

Error Types

In Java you can find two types of error: Compile Time Error, and Run Time Error. But for Python, only the Traceback or Runtime error is found.

Speed

Java is comparatively faster since it's a compiled language and as a result takes less time to run. Python, on the other hand is an interpreted language and is dynamically typed and hence a little slower. But there are instances where one outperforms the other.

First Class Objects:

In Python everything is a first class object, including functions and classes, whereas functions cannot be passed as arguments in JAVA and are not first class objects.

Java outperforms python on several instances in terms of speed.

Python has a shorter learning curve and is easier to understand in contrast to JAVA.

Hello world in Java

```
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello World");
}
}
```

Hello world in Python

```
print ('Hello, World!')
```

CLASS

Code Snippets: Classes

Java

```
public class Student {

        String name;
        int age;
        String major;
        public Student() {
        }
        public Student(String name, int age, String major) {
                this.name = name;
                this.age = age;
                this.major = major;
        }
}
```

Python

```
class Student:
    def __init__(self, name, age, major):
        self.name = name
        self.age = age
        self.major = major
    def is_old(self):
        return self.age > 100
```

Difference/Commonlaity

- Constructors in both languages are written in different way, in Python its with __init__ whereas in Java it's a method with the name same as that of the class.
- The indentation in Python is also a difference with respect to the Java code where indentation doesn't carry any meaning.
- There is no universally-accepted Python convention for naming classes whereas class names begin with an uppercase letter
- Pure functions which cannot be accessed through class instance and must be made static in Java, whereas a decorator can be used in Python for the same.
- Java uses this while python uses self to access the current instance.

Advantages/Disadvantages

- The code is more verbose in Java in comparison to Python for the same representation as shown above.

MULTIPLE INHERITANCE AND MULTIPLE INHERITANCE RESOLUTION:

Code Snippets: Multiple Inheritance

Java

Java **doesn't** allow multiple inheritance for classes; Java has interfaces for this purpose.

```
public interface InterfaceA {

        public void doSomething();
}
public interface InterfaceB {

        public void doSomething();
}

public interface InterfaceC extends InterfaceA, InterfaceB {

        //same method is declared in InterfaceA and InterfaceB both
        public void doSomething();

}
```

Python

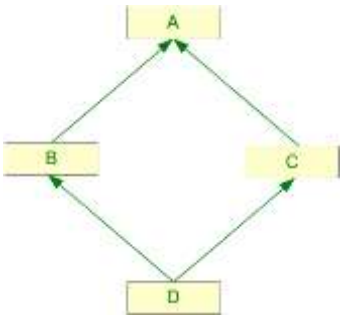
```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```

Multiple inheritance makes it easier to compose classes from small base classes that implement functionality and have properties to remember state. When done right, you can get a lot of reuse of small code without having to copy-and-paste similar code to implement interfaces. The main con is that if two classes have a method with the same name, the new subclass doesn't know which one to call.

The Diamond Problem or the, deadly diamond of death":



The "diamond problem" (sometimes referred to as the "deadly diamond of death") is the generally used term for an ambiguity that arises when two classes B and C inherit from a superclass A, and another class D inherits from both B and C. If there is a method "m" in A that B or C (or even both of them) have overridden, and furthermore, if does not override this method, then the question is which version of the method does D inherit? It could be the one from A, B or C

Multiple Inheritance Resolution:

When implementing multiple inheritance, Python builds a list of classes to search as it needs to resolve which method must be called when one is invoked by an instance. As the name suggests, the method resolution order will search the depth-first, then go left to right.

Python

```
class A:
        def whereiam(self):
                print "I am in A"


class B(A):
        def whereiam(self):
                print "I am in B"


class C(A):
        def whereiam(self):
                print "I am in C"


class D(B, C):
    pass
```

So, the resolution order in the above example will be D, B, A, C, A.

TYPE():

Code Snippets:

Java

Since Java has Primitive Data Types, Classes and Objects, there are different ways to us type in all of them. Following are some ways:

```
//Get type of Object
com.mypackage.MyClass c = new com.nyu.MyClass();
System.out.println(c.getClass().getName());
Output:com.nyu.MyClass

//Get type of primitive
float f = "1.23";
String type = ((Object)f).getClass().getName();
System.out.println(type);
Output:java.lang.Float
```

Python :

Since everything in Python is object the type(object) returns the class type of the argument passed in the method. The type function accepts single or three parameters. If single argument type(obj) is passed, it returns the type of given object. If three arguments type(name, bases, dict) is passed, it returns a new type object.

```
print(type([]) is list)

Output: True

class myclass:
    print "hello"

print type(myclass)
Output: <type 'classobj''
```

ISINSTANCE ():

Code Snippets:

Java

isInstance() in Java is used to check the class of object at runtime.

```
public static boolean fun(Object obj, String c)
                    throws ClassNotFoundException
    {
        return Class.forName(c).isInstance(obj);
    }
public static void main(String[] args)
                    throws ClassNotFoundException
    {
        Integer i = new Integer(5);

        boolean b = fun(i, "java.lang.Integer");

        // print false as i is not instance of class
        // String
        boolean b1 = fun(i, "java.lang.String");

        System.out.println(b);
        System.out.println(b1);
    }
Output: true
        false
```

Python :

In Python, isinstance() function checks if the object (first argument) is an instance or subclass of classinfo class (second argument).

```
class Foo:
  a = 5

fooInstance = Foo()

print(isinstance(fooInstance, Foo))
print(isinstance(fooInstance, (list, tuple)))
print(isinstance(fooInstance, (list, tuple, Foo)))
print type(myclass)
Output: <type 'classobj''
```

CLASS METHODS:

Code Snippets:

Java

Class methods are methods that are called on the class itself, not on a specific object instance. In Java the static modifier ensures implementation is the same across all class instances. The keyword static is used to represent a class method/static method.

```
public class Numbers {
    public static int dm(int number1, int number2) {
        int minimum = number2;
        if (number1 < number2) minimum = number1;
        return minimum;
    }
}

//Calling from main method
//calling method outside of the class
int min = Numbers.min(3, 5);
//this would NOT work, static methods cannot be accessed with an object
min = numberClass.min(3, 5);
```

Python :

In Python, the concept is different as the classes can be passed as arguments to the function. Further, a **class method** receives the class as implicit first argument, just like an instance method receives the instance. It can modify a class state that would apply across all the instances of the class. For example it can modify a class variable that will be applicable to all the instances whereas we need to use Reflections in Java to modify the class state.

```
class C(object):
    @classmethod
    def fun(cls, arg1, arg2, ...):
       ....
fun: function that needs to be converted into a class method
returns: a class method for function.
```

Static Methods in Python:

The Static method in Python doesn't receive an implicit first argument as opposed to a class method. As a result, a static method cannot access or modify the state of the class.

REFLECTION/INTROSPECTION:

Code Snippets: type

Java

In Java, reflection allows your code to interrogate a class or instance to get the properties at run time.

```
//Get type of Object
com.mypackage.MyClass c = new com.nyu.MyClass();
System.out.println(c.getClass().getName());
Output:com.nyu.MyClass

//Get type of primitive
float f = "1.23";
String type = ((Object)f).getClass().getName();
System.out.println(type);
Output:java.lang.Float
```

Python:

The equivalent term for reflection in Python is Introspection. However, Python isn't strictly typed so there isn't really a notion of self-describing types. That means that some of the things we typically use reflection for entity models and custom serialization don't translate well.

```
#define a class
class Foo:
          def bar(self):
                    return 123

 #get the list of methodsmethodNames = filter(lambda x: not x.startswith('_'),
dir(foo.__class__))

 #call the first method
getattr(foo, methodNames[0])()
```

OBJECT AND CLASS ENCAPSULATION:

Code Snippets: type

Java

In encapsulation, the variables or data of a class is hidden from any other class and can be accessed only through any member function of own class in which they are declared. The data in a class is hidden from other classes, so it is also known as data-hiding. Encapsulation can be achieved in Java by: Declaring all the variables in the class as private and writing public methods in the class to set and get the values of variables as shown below

```
public class Encapsulate
{
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String name;
    private int roll;
    private int age;

    // get method for age to access
    // private variable age
    public int getAge()
    {
      return age;
    }

    // get method for name to access
    // private variable name
    public String getName()
    {
      return name;
    }

    // get method for roll to access
    // private variable roll
    public int getRoll()
    {
       return roll;
    }

    // set method for age to access
    // private variable age
    public void setAge( int newAge)
    {
      age = newAge;
    }

    // set method for name to access
    // private variable name
    public void setName(String newName)
    {
      name = newName;
    }

    // set method for roll to access
    // private variable roll
    public void setRoll( int newRoll)
```

```
    {
      roll = newRoll;
    }
}
```

Python :

We can restrict access to methods and variables. This can prevent the data from being modified by accident and is known as encapsulation. The following example shows encapsulation in Python

Private methods:

We create a class Car which has two methods: drive() and updateSoftware(). When a car object is created, it will call the private methods __updateSoftware().

```
#!/usr/bin/env python

class Car:

def __init__(self):
self.__updateSoftware()

def drive(self):
print 'driving'

def __updateSoftware(self):
print 'updating software'

redcar = Car()
redcar.drive()
#redcar.__updateSoftware()  not accesible from object.
class
```

Private variables

A private variable can only be changed within a class method and not outside of the class.

```
#!/usr/bin/env python

class Car:

__maxspeed = 0
__name = ""

def __init__(self):
self.__maxspeed = 200
self.__name = "Supercar"

def drive(self):
print 'driving. maxspeed ' + str(self.__maxspeed)

redcar = Car()
redcar.drive()
redcar.__maxspeed = 10  # will not change variable because its private
redcar.drive()
```

Type	Description
public methods	Accessible from anywhere
private methods	Accessible only in their own class. starts with two underscores
public variables	Accessible from anywhere
private variables	Accessible only in their own class or by a method if defined. starts with two underscores

References:

- https://www.geeksforgeeks.org/class-method-vs-static-method-python/
- https://anh.cs.luc.edu/363/notes/JavaVsPython.html#First
- https://syntaxdb.com/ref/java/class-methods
- https://stackoverflow.com