Groovy

Dynamic typing

Types in Java – abstract methods

```
public void takeHelp(Man man) {
  //...
  man.helpMoveThings();
  //...
}

public abstract class Human {
  public abstract void helpMoveThings();
  //...
}

public void takeHelp(Human human) {
  //...
  human.helpMoveThings();
  //...
}
```

Types in Java – interfaces

```
public interface Helper {
  public void helpMoveThings();
}

public void takeHelp(Helper helper) {
  //...
  helper.helpMoveThings();
  //...
}
```

Duck typing

```
def takeHelp(helper) {
  //...
  helper.helpMoveThings()
  //...
}
```

- Design by capability.
  - Instead of asking the helper to conform to some explicit interface, we’re making use of the object’s capability, relying upon an implicit interface.
- Duck typing - ”if it walks like a duck and quacks like a duck, it must be a duck.
- Example 00

Problems with dynamic typing

- We might mistype the method name when creating one of the helpers.
  - Unit testing
- Without the type information, how do we know what to send to the method?
  - Naming conventions
- What if we send the method a nonhelper (an object that's not capable of moving stuff)?
  - Method discovery

Polymorphism in Java

```
public class Employee {
  public void raise(Number amount) {
    System.out.println("Employee got raise");
  }
}

public class Executive extends Employee {
  public void raise(Number amount) {
    System.out.println("Executive got raise");
  }
  public void raise(java.math.BigDecimal amount) {
    System.out.println("Executive got outlandish raise");
  }
}

public class GiveRaiseJava {
  public static void giveRaise(Employee employee) {
    employee.raise(new BigDecimal(10000.00));
  }
  public static void main(String[] args) {
    giveRaise(new Employee());
    giveRaise(new Executive());
  }
}
```

Multimethods in Groovy

```
void giveRaise(Employee employee) {
  employee.raise(new BigDecimal(10000.00))
  // same as
  //employee.raise(10000.00)
}
giveRaise new Employee()
giveRaise new Executive()
```

- Groovy picks the correct implementation based not only on the target object, but also on the parameter(s) we send to the call.
- Since the method-dispatching is based on multiple objects - the target plus the parameters - this is called multiple dispatch or multimethods