Method dispatch

Multiple Dispatch nebo také MultiMethods je jedna z vlastností některých OO programovacích jazyků, v nichž může být funkce nebo metoda dynamicky zavolána/dispatched na základě typu až při run time. Podobnost můžeme vidět z přetěžování funkcí (overloading), kdy je vše založeno na statických typech.

Vysvětlení dispatchingu
Vývojáři typicky organizují bloky svého kódu do podprogramů, procedur, funkcí nebo metod. Název obvykle závisí na daném jazyce a někdy se liší i svými vlastnostmi. Jméno funkce se používá jako odkaz při jejím volání. V průběhu vykonávání kódu zavolání funkce způsobuje předávání řízení volané funkci, řízení se pak často vrací volajícímu.

Jména funkcí často charakterizují jejich činnost a občas se stane, že bychom chtěli několika funkcím přiřadit stejné jméno, přičemž často dělají něco podobného, ale operují nad jinými typy vstupních dat. V takovém případě už není název funkce dostatečný identifikátor, který určuje, která část kódu se má vykonat. Místo jména se využívá i počtu a typu argumentů, díky čemuž můžeme vybírat z několika implementací funkce.

V tradičních (*single dispatch*) OO jazycích když zavoláme metodu ("send a message" ve SmallTalku, "call a member function" v C++), s jedním z jejích argumentů se zachází trochu jinak a je používán k určení metody toho daného jména.

V jazycích s *multiple dispatch* je se všemi argumenty zacházeno souměrně při výběru metody, porovnání první, druhé, třetí atd. pozice uvnitř dané syntaxe.

Common Lisp Object System (CLOS) je typickým (a brzkým) případem *multiple dispatch*.

Datové typy
Pokud pracujeme s jazyky, které umí rozlišovat datové typy během kompilace, může dojít k výběru mezi alternativami právě až v době kompilace. Oproti tomu u jazyků, které odkládají identifikaci typů až na run time, dochází k výběru při run time.

Příklady:
Příklady pro některé jazyky zde:
http://en.wikipedia.org/wiki/Multiple_dispatch#Examples

Nebo si představme, že máme problém zahrnující několikanásobné typy/třídy a rozhodneme se využít overloadingu v Javě. Implementujeme pouze jednu metodu pro každou z tříd a je to! Dvě podtřídy třídy:

```
Person:
void doStuff(Worker worker) {
   ...
}
void doStuff(Capitalist capitalist) {
   ...
}
```

A nyní, pokud zavoláme doStuff nějak takto:

```
Person person = ...
x.doStuff(person);
```

Zdá se, že by to mělo fungovat, mělo by to zavolat správnou metodu v závislosti na tom,

které třídy `person` je. Ale takto to nelze! Volání metod v Javě musí znát typy svých argumentů už při kompilaci. Musíme castnout (přetypovat) `person` na jednu z podtříd, jinak si začne překladač stěžovat, že neví, kterou má použít a nemůže vám pomoci, ale vy to přeci víte ;-)

Příklad na dynamický method dispatch v Javě:

```
// Dynamic Method Dispatch
class A {
      void callme() {
            System.out.println("A");
      }
}

class B extends A {
      // override callme()
      void callme() {
            System.out.println("B");
      }
}

class C extends A {
      // override callme()
      void callme() {
            System.out.println("C");
      }
}

class Dispatch {
      public static void main(Strings args[]) {
            A a = new A(); // object of type A
            B b = new B(); // object of type A
            C c = new C(); // object of type A
            A r;              // obtain a reference of type A

            r = a; // r refers to an A object
            r.callme(); // calls A version of callme

            r = b; // r refers to an B object
            r.callme(); // calls B version of callme

            r = c; // r refers to an C object
            r.callme(); // calls C version of callme
      }
}
```

Výstup bude:

```
A
B
C
```

Tento program vytvoří jednu nadtřídu (superclass) nazvanou `A` a dvě podtřídy (subclasses) `B` a `C`. Tyto podtřídy překryjí (override) metodu `callme()` deklarovanou v `A`. V metodě `main()` jsou deklarovány objekty typů `A`, `B` a `C`, rovněž také reference na `A`, nazvaná `r`. Program poté přiřadí `r` reference ke každému typu objektu a používá reference k vykonání `callme()`. Jak dokazuje výstup, vykonání `callme()` je rozlišeno v závislosti na typu objektu, který je referován (odkazován) v době volání.

Nebo stejný příklad:
Máme třídu a podtřídu, obě mají metodu stejně pojmenovanou se stejnými parametry. Na instance podtřídy ukazujeme pomocí reference supertřídy. Kterou metodu zavolat?

```
class A {
      void call() {
            print "Superclass"
      }
}
class B extends A {
      void call() {
            print "subclass"
      }
}

A a = new A();
B b = new B();
A r = a;
r.call(); // vypise se superclass, protoze v r je objekt A tridy
r=b;
r.call(); // vypise se subclass, protoze v r je objekt B tridy
```

To, co bude v `r`, je známo až za běhu. Rozhoduje se tedy podle toho, co je v referenci skutečně za objekt, ne podle typu objektu.