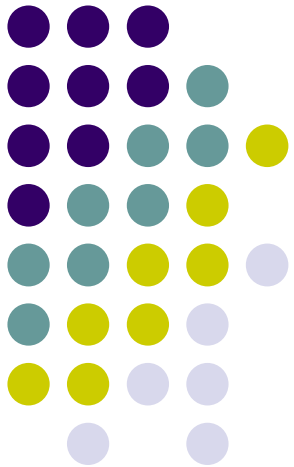
Computação Paralela

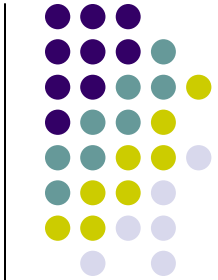
Introdução ao AspectJ

João Luís Ferreira Sobral
Departamento do Informática
Universidade do Minho

Novembro 2006



Introdução ao AspectJ

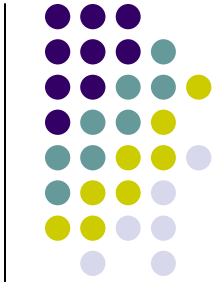


Concebido para lidar com as facetas transversais do código

- Exemplos: *logging*, *timing*

```
public class PrimeFilter {

    int[] myPrimes;     // buffer to hold primes already calculated
    int nPrimes;        // number of primes calculated
    double start;       // time account
    int cPrimes;                                    // local primes count
    int cPack=0;        // current received pack
    int myNum = (int) (100*Math.random());


    // initialize filter parameters
    public void init(int myMinP,int myMaxP) {
       nPrimes=0;
       cPack=0;

       // calculate primes between myMinP and myMaxP
       int[] pr = new int[myMaxP-myMinP];
       int nl = PrimeCalc.lowPrimes(myMaxP,pr);

       myPrimes = new int[nl];
       for(int i=0; i<nl; i++)
          if (pr[i]>=myMinP && pr[i]<=myMaxP) myPrimes[nPrimes++]=pr[i];
       System.out.println(myNum + ": " + nPrimes + " primes " + myMinP + "-" + myMaxP );
       cPrimes=0;
       // starts time account
       start = new Date().getTime();
    }
```

```
    // process pack of numbers
    public int[] filter(int[] num) {
        cPack++;

      // remove nom primes from the list
      for(int i=0; i<num.length; i++) {
         if (num[i]>2) {    //
                              if (PrimeCalc.isPrime(num[i],myPrimes,nPrimes) )
                                                  cPrimes++;
                              else num[i]=0;
         }
      }
      // shows processing time
      double end = new Date().getTime();
      System.out.print(myNum + ": " + cPack + " - Time " + (end - start) + "ms");
      System.out.println(" - " + cPrimes + " primes");

      return(num);
    }
}
```



Introdução ao AspectJ

Concebido para lidar com as facetas transversais do código

- Static crosscutting – permite alterar a hierarquia de tipos

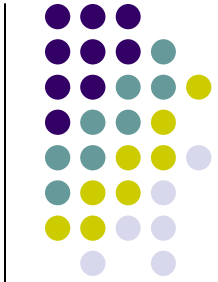
```
public aspect StaticIntroduction {
        declare parents: Point implements Serializable;
        public void Point.migrate(String node) {  System.out.println("Migrate to node" + node); }
}
```

- Dynamic crosscutting – permite alterar o comportamento dinâmico da aplicação

```
public aspect Logging {
        void around(Point obj, int disp) : call(void Point.move*(int)) && target(obj) && args(disp) {
                System.out.println("Move called: target object = " + obj + " Displacement " + disp);
                proceed(obj,disp);                                              // proceed the original call
        }
}
```

```
public class Point {

        private int x=0;

        private int y=0;

        public void moveX(int delta) {  x+=delta; }

        public void moveY(int delta) {  y+=delta; }

        public static void main(String[] args) {

                Point p = new Point();

                p.moveX(10);

                p.moveY(5);
        }

}
```

- Os aspectos são semelhantes a classes, podendo conter métodos, variáveis de instância
- Só existe uma instância de cada aspecto por aplicação
  - **Frequentemente o aspecto deve gerir múltiplas instancias do mesmo mecanismo**



Introdução ao AspectJ

Exercícios

- Altere o exemplo da aula da semana 5 (exemplo dos primos) por forma a que o *logging* e *timing* sejam implementados por aspectos (ver também acetato 2 desta aula)