p1 q3 4
ACN

P1 Q4

# 1 Java half 2000

The following attempt at Java code might have been written by a beginner. Identify (but do not correct) as many of the mistakes as you can. Explain how each oddity you identify is either something that would prevent the program from compiling, something that would cause it to stop abruptly reporting failure at runtime, a reason why the program might not do anything sensible or just a stylistic oddity.

```
//* A comment to start with: Exam:2000 */
public Class mycode.java
{
void static public fun main(String [junk])   {
  begin
    ~~Leaf tr = null~~;
int i   for (i=1; i>10; ++i) tr = new Node(i, tr)   type
    tr.print();              typechecking
  end;
}

class Leaf
{
    int~~eger~~ value;
    Leaf(int value)
    {   this = value; }      this.value = value;
    public void print()
    {   System.out.println(value); }
}                            + "" cast to string

class Node extends Leaf
{
    Leaf left, right;
    Node(leaf l, Leaf r)
    {   left = l, right = r;
    }
    void print()                          no base case
    {   left.print();
        System.out.println("val=" @ value);
        right.print();
    }
}
```

[10 marks given as proportion of total errors found and clearly explained]

## 1.1 Marking notes

```
This code is a MESS. It uses the wrong keywords in loads
of places (eg Class for class, "fun" in declaration of
main, bad syntax for array of strings arg. The class
constructors for the two sub-classes are not as used in
the main program. The print method in the sub-class is
```

there are a few more than 20 errors in all.
or
infelicities

more restrictive in access than in the parent. Etc Etc!! The "//* ... */" comment at the head is valid but looks wrong. The print method (such as it is) would bard when it hits a null reference. LOTS for people to spot and a proper marking scheme grades in part on which cases turn out in reality to be easy for people to spot and on clarity of explanation.

## 2 Java full

Write Java classs that provides support for arithmetic on the integers worked with relative to some prime modulus $p$. An instance of the class should be constuctable specifying the modulus, and then it should provide methods to create numbers and add, subtract, multiply, divide and print them. Note that the Discrete Mathematics lectures explained about arithmetic modulo a prime, and that the reciprocal of a number (a say) (mod $p$) can be found by solving the equation

$$ab = 1 \bmod p$$

As a sample of the desired behaviour for your class, here is some test code for it:

```
Modular d = new Modular(7);  // work mod 7
ModInt a = d.reduceMod(10);  // create "10 mod 7"
ModInt b = d.reduceMod(20);  // create "20 mod 7"
ModInt c = a.divide(b);      // work out a/b mod 7
c.print();
```

Note that I am suggesting a class called Modular that keeps track of the modulus $p$, and a second class ModInt to stand for numbers: these are created for the user via a method in Modular.

Your code should compain in some manner if an attempt is made to (say) add a number that is defined modula 7 to one that is defined modulo 11.

*[20 marks]*

### 2.1 Marking notes

```
class Modular
{
int p;
Modular(int p)
{ this.p = p; }

ModInt reduceMod(int n)
{
    return new ModInt(n%p, p);
}
}

class ModInt
{
```