Ejercicio 2

```
#include <iostream>
using namespace std;

class Padre {
protected:
    int atr1, atr2;
public:
    Padre(int a1, int a2) : atr1(a1), atr2(a2) {}

    virtual int metodo1() const {
        return atr1 + atr2;
    }
};

class Hija : public Padre {
public:
    Hija(int a1, int a2) : Padre(a1, a2) {}

    int metodo1() const override {
        return atr1 * atr2;
    }
};

template <typename T>
class PadreGenerico {
protected:
    T atr1, atr2;
public:
    PadreGenerico(T a1, T a2) : atr1(a1), atr2(a2) {}

    virtual T metodo1() const {
        return atr1 + atr2;
    }
};

template <typename T>
class HijaGenerica : public PadreGenerico<T> {
    using PadreGenerico<T>::atr1;
    using PadreGenerico<T>::atr2;
public:
    HijaGenerica(T a1, T a2) : PadreGenerico<T>(a1, a2) {}

    T metodo1() const override {
        return atr1 * atr2;
    }
};
```

```
int main() {
    Padre p(5, 6);
    Hija h(5, 6);
    cout << "No genéricas" << endl;
    cout << p.metodo1() << "\t" << h.metodo1() << endl;
    PadreGenerico<int> pg(5, 6);
    HijaGenerica<double> hg(5.5, 6.2);
    cout << "Genéricas" << endl;
    cout << pg.metodo1() << "\t" << hg.metodo1() << endl;
}
```

Apartado a)
El polimorfismo por inclusión se puede observar en la redefinición del método "metodo1" por parte de la clase Hija.

Apartado b)
En el caso de que no existiera el polimorfismo por inclusión sería necesario introducir un nuevo método para poder realizar la operación realizada en el "metodo1".

Apartado c)
En el uso de los tipos de datos T que representan tipos de datos desconocidos.

Apartado d)
Si no existiera sería necesario realizar una clase para cada tipo de dato que se quiera representar.

Ejercicio 3

```
type Nombre = String
type Tamagno = Int
data ArbolDirectorios = Fichero Nombre Tamagno | Directorio Nombre
ArbolDirectorios ArbolDirectorios
                    deriving (Eq,Ord,Show,Read)

-- Función auxiliar para buscar si existe el nombre de un directorio o
fichero en la ruta
existeDirectorio :: ArbolDirectorios -> String -> Bool
existeDirectorio (Fichero nom tam) x = nom == x
existeDirectorio (Directorio nom i d) x = existeDirectorio i x ||
existeDirectorio d x || x == nom

buscarRuta :: ArbolDirectorios -> [String] -> Bool
buscarRuta (Fichero nom tam) [] = False
buscarRuta (Directorio nom i d) [] = False
buscarRuta (Fichero nom tam) xs
    | (length xs == 1) && nom == head xs = True
    | otherwise = False
buscarRuta (Directorio nom i d) xs
    | length xs == 1 = nom == head xs
    | nom == head xs = buscarRuta d (tail xs) || buscarRuta i (tail xs)
    | otherwise = False

stat :: ArbolDirectorios -> [String] -> Int
stat (Fichero nom tam) [] = tam
stat (Fichero nom tam) xs
    | nom == head xs = tam
    | otherwise = 0
stat (Directorio nom iz de) xs
    | buscarRuta iz (tail xs) = stat iz (tail xs)
    | otherwise = stat de (tail xs)


escribirRuta :: ArbolDirectorios -> String -> [String]
escribirRuta (Fichero nom tam) n = [nom]
escribirRuta (Directorio nom i d) n
    | nom == n = [nom]
    | existeDirectorio i n = nom : escribirRuta i n
    | existeDirectorio d n = nom : escribirRuta d n
```

```
find :: ArbolDirectorios -> [[String]]
find (Fichero nom tam) = [[nom]]
find (Directorio nom i d) = ([escribirRuta actual nom] ++ find i) ++
([escribirRuta actual nom] ++ find d)
    where
        actual = Directorio nom i d

arbol :: ArbolDirectorios
arbol = Directorio "/" (Directorio "usr" (Directorio "bin" (Fichero
"g++" 100)(Fichero "vi" 200)) (Fichero "" 0)) (Directorio "sbin"
(Fichero "ghc" 50) (Fichero "" 0))
```

Ejercicio 4

```
#include <list>
#include <iostream>
using namespace std;
class Point {
    double x, y;
public:
    Point(double x, double y) : x(x), y(y) {}

    double suX() const {
        return x;
    }

    double suY() const {
        return y;
    }

    bool operator<(Point a) {
        return (x < a.suX() || y < a.suY());
    }
};

class Objeto {
protected:
    list<Point> puntos;
public:
    Objeto(list<Point>& p) : puntos(p) {}

    list<Point> points() const {
        return puntos;
    }

    virtual bool contains(Point x) const {
        bool estaContenido = true;
        for (auto i : puntos) {
            if (i < x) {
                estaContenido = false;
            }
        }
        return estaContenido;
    }
};
```

```
class Circle : public Objeto {
    Point centro;
    int radio;
public:
    Circle(const Point& cen, int rad, list<Point> p) : centro(cen),
radio(rad), Objeto(p) {}
};

class Rectangle : public Objeto {
    Point izquierda, derecha;
public:
    Rectangle(const Point& izq, const Point& der, list<Point> p) :
izquierda(izq), derecha(der), Objeto(p) {}
};

class Polygon : public Objeto {
public:
    Polygon(list<Point> p) : Objeto(p) {}
};

bool collide(const Objeto& a,const  Objeto& b) {
    list<Point> la = a.points();
    list<Point> lb = b.points();
    for (auto i : la) {
        if (b.contains(i)) {
            return true;
        }
    }
    for (auto i : lb) {
        if (a.contains(i)) {
            return true;
        }
    }
    return false;
}

bool collide_any(const Objeto& objeto,const list<Objeto*> lista) {
    for (auto i : lista) {
        if (collide(objeto, *i)) {
            return true;
        }
    }
    return false;
}
```