Nombre: ..... Fecha: / 12 / 2008

HOJA DE EJERCICIOS 3
HERENCIA ENTRE CLASES Y POLIMORFISMO DE MÉTODOS

Esta hoja de ejercicios nos servirá para repasar las nociones más importantes de los Temas 2 y 3.

1. Los dos requisitos previos fundamentales para que podamos hablar de que un método se comporta de modo polimorfo son:

1.

2.

2. EL polimorfismo de métodos está directamente vinculado con el enlazado dinámico de métodos. Explica cuál es el comportamiento por defecto en Java y C++ con respecto al enlazado:

C++:

Java:

3. Supongamos que un determinado método “redefinido(): void” aparece en dos clases, clase “Base” y clase “Derivada”. Supón que nuestro compilador sólo ofrece enlazado estático ¿Qué versiones del método se llamaría en el siguiente fragmento?

```
Base obj1 = new Derivada (“un objeto”);
Derivada obj2 = new Derivada (“otro objeto”);

//Invocación 1:
obj1.redefinido ();
//Invocación 2
obj2.redefinido ();
//Invocación 3
obj1 = obj2;
obj1.redefinido();
```

Invocación 1:

Invocación 2:

Invocación 3:

4. Supongamos que en ejemplo anterior nuestro compilador ofreciera enlazado dinámico ¿Qué versiones de los métodos serían invocadas?

```
Base obj1 = new Derivada (“un objeto”);
Derivada obj2 = new Derivada (“otro objeto”);

//Invocación 1:
obj1.redefinido ();
//Invocación 2
obj2.redefinido ();
//Invocación 3
obj1 = obj2;
```

obj1.redefinido();

Invocación 1:

Invocación 2:

Invocación 3:

5. Repasemos la diferencia entre declaración y construcción. Tenemos una clase “Base” y una clase “Derivada” que hereda de la primera ¿Qué sucedería al compilar el siguiente fragmento de código? Relaciona tu repuesta con la idea de subtipado.

```
Base obj1 = new Derivada ();
Derivada obj2 = new Derivada ();

obj2 = obj1;
```

6. ¿Qué sucedería al cambiar el orden de la asignación anterior? Relaciona tu repuesta con la idea de subtipado.

```
Base obj1 = new Derivada ();
Derivada obj2 = new Derivada ();

obj1 = obj2;
```

Pasamos ahora a los lenguajes Java y C++.

7. ¿Qué tipo de enlazado ofrece C++ por defecto? (Estático o dinámico) ¿Hay polimorfismo de métodos por defecto?

8. ¿Qué tipo de enlazado ofrece C++ por defecto? (Estático o dinámico) ¿Hay polimorfismo de métodos por defecto?

9. ¿Cuáles son las dos condiciones necesarias para conseguir enlazado dinámico de métodos en C++?

1.

2.

10. Repaso sobre herencia: ¿Cuál es la primera orden que debe aparecer siempre en un constructor de una clase derivada, tanto en Java como en C++?

11. Observa el siguiente fragmento de código en C++. Supón que existe un método "redefinido(): void" que aparece tanto en las clases "Base" como en la clase "Derivada" ¿Qué versión del método será invocada?

```
Base obj1;
Derivada obj2;

obj1 = obj2;
obj1.redefinido();
```

12. Imagina el anterior fragmento de código en Java ¿Qué sucedería?

```
Base obj1;
Derivada obj2 = new Derivada();

obj1 = obj2;
obj1.redefinido();
```

13. Supongamos que el método "redefinido(): void" aparece ahora en el fichero "Base.h" con el modificador "virtual" ¿Qué versión del mismo sería llamada?

```
class Base{

private: char nombre[20];
public:

Base();
virtual void redefinido();

}

class Derivada: public Base {

public:
Derivada();
void redefinido();

}

int main(){

Base * obj1;
Derivada * obj2 = new Derivada;
```

```
obj1 = obj2;
obj1->redefinido();

return 0;
}
```

¿Se cumplen los requisitos para que haya polimorfismo (y enlazado dinámico)? ¿Qué versión del método “redefinido(): void” será invocada?

14. Pasemos a ver una nueva situación en la cual nos gustaría contar con enlazado dinámico: trabajo con arrays (o estructuras genéricas).

Supongamos que tenemos la misma declaración de clases que en el ejercicio 13.

¿Qué versiones del método “void: redefinido()” se invocarían en el siguiente contexto?

```
int main(){

        Base * obj1 =new Base;
        Derivada * obj2 = new Derivada;

        Base array_base [2];
        array_base [0] = (*obj1);
        array_base [1] = (*obj2);

        for (int i = 0; i < 2; i++){
                //Invocación “i”:
                array_base[i].redefinido();
        }

        return 0;
}
```

Invocación 1:

Invocación 2:

15. Repetimos el ejercicio anterior con ligeras modificaciones (recuerda que el método “redefinido(): void” tiene el modificador “virtual”.

Supongamos que tenemos la misma declaración de clases que en el ejercicio 13.

¿Qué versiones del método “void: redefinido()” se invocarían en el siguiente contexto?

```
int main(){

        Base * obj1 =new Base;
        Derivada * obj2 = new Derivada;

        Base * array_base [2];
        array_base [0] = obj1;
        array_base [1] = obj2;

        for (int i = 0; i < 2; i++){
```

```
        //Invocación “i”:
        array_base[i]->redefinido();
    }

    return 0;
}
```

Invocación 1:

Invocación 2:

16. Incidimos una vez más en la idea de subtipado ¿Qué resultado obtendrías el compilar el siguiente fragmento de código en C++? (El resultado sería equivalente en Java, la respuesta está directamente relacionada con la relación de subtipado)

```
int main(){

    Base * obj1 =new Derivada;
    Derivada * obj2 = new Derivada;

    Derivada * array_derivada [2];
    array_derivada[0] = obj1;
    array_derivada[1] = obj2;

    for (int i = 0; i < 2; i++){
        //Invocación “i”:
        array_derivada[i]->redefinido();
    }

    return 0;
}
```

Invocación 1:

Invocación 2:

17. Repasamos también la forma de obtener polimorfismo en funciones auxiliares en C++ ¿Qué versión del método “redefinido(): void” se invocaría a través de la función “auxiliar (Base): void” en el siguiente contexto?

```
void auxiliar (Base);

int main(){

    Base * obj1 =new Derivada;
    Derivada * obj2 = new Derivada;

    //Invocación 1:
    auxiliar (*obj1);
    //Invocación 2:
    auxiliar (*obj2);
    return 0;
}

void auxiliar (Base bas){
```

```
        bas.redefinido();
}
```

Invocación 1:

Invocación 2:

18. ¿Qué versión del método “redefinido(): void” se invocaría a través de la función “auxiliar(Base *): void” en el siguiente contexto?

```
void auxiliar (Base *);

int main(){

        Base * obj1 =new Derivada;
        Derivada * obj2 = new Derivada;

        //Invocación 1:
        auxiliar (obj1);
        //Invocación 2:
        auxiliar (obj2);
        return 0;
}

void auxiliar (Base * bas){
        bas->redefinido();
}
```

Invocación 1:

Invocación 2:

19. Veamos ahora una de las propiedades del modelo de memoria de Java: paso por referencia de parámetros.

¿Qué sucedería al realizar las siguientes acciones en Java?

```
public class principal{

public static void main (String [] args){

        Base obj1 = new Base();
        Derivada obj2 = new Derivada();

        permutar (obj1, obj2);
        //Invocación 1
        obj1.redefinido();
        //Invocación 2
        obj2.redefinido()

}

public static void permutar(Base bas1, Base bas2){

        Base aux;
```

```
        aux = bas1;
        bas1 = bas2;
        bas2 = aux;
        }
}
```

Invocación 1. ¿Qué método “redefinido (): void” ha sido invocado?

Invocación 2. ¿Qué método “redefinido (): void” ha sido invocado?