Programación II
Clases paramétricas<G>

Definición

Un TAD o clase paramétrica recibe el tipo(o género G) de clase como parámetro formal. Se utiliza para generalizar el comportamiento de un TAD T, y tiene varias consecuencias inmediatas:
-Mayor abstracción
-Reutilización de código

Si G es un objeto predefinido de java(Integer, String, etc) , generalmente funcionan todas las operaciones de G.
Si G es un objeto definido por el usuario es necesario también redefinir ciertas funciones para no obtener comportamientos no deseados:
-Equals()
-HashCode()

Todo TAD[1] o Objeto de java la tiene definidas por herencias. Se trata de redefinir el comportamiento por el semánticamente correcto.

Sintaxis y semántica

Vamos a mostrar un ejemplo de cómo hacer una clase paramétrica y analizaremos los resultados del método imprimir.

```
Class Ejemplo<G>
G x;
G y;
Public Ejemplo(G valor){
        X = valor;
        Y  = valor;
}
public  String imprimir(){
        return  x.toString() + y.toString();  //implementación1
}
```

[1] Todo TAD de java es un Objeto. Object implementa Equals y HashCode. Por lo tanto es necesario redefinir dichas clases para no tener comportamientos inesperados.

Main

```
Ejemplo ej1 = new Ejemplo(7)<Integer >;
Ejemplo ej2 = new Ejemplo("7")<String>;

System.out.println(Ej1.imprimir());
System.out.println(Ej2.imprimir());
```

Ejercicio 1

Implementar la clase Ejemplo y ver que se imprime para cada ejemplo(ej1 y ej2).
Por una limitación de Java, no poder convertir T a String, la siguiente expresión no compila:

return (String)(x + y); //**implementación2**

Sin embargo, analizar que pasaría si imprimir() utilizara la **implementación2**.

Análisis

Para el ejemplo1, la suma es de enteros, mientras que para el ejemplo2 la suma es de Strings.
El cast a string "(String)" es necesario para cumplir con el género de imprimir().

Extendiendo G para que sea comparable

En algunos casos, necesitamos que G sea comparable.
Por ejemplo podríamos querer generalizar una agenda, donde G representa a una persona de la agenda.
La forma de declarar esa intención en java es agregando "extends Comparable<G> ":

```
public class Agenda<P extends Comparable<P>> {
```

Luego, en la implementación, se podría se la siguiente manera:

```
public P devolverLaPersonaMaschica (P persona1, P persona2) {
	P ret;
	if (persona1.compareTo(persona2) == 1) {    // persona1 < persona2
		ret = persona1;
	}else{
		ret = persona2;
	}
	return ret;
}
```

Donde la clase persona, tiene

```
public class Persona implements Comparable<Persona> {

	private Integer dni;

	@Override
	public int compareTo(Persona p){
		int ret = 0;
		if (dni == p.dni){ret= 0;}
		if (dni < p.dni){ret= -1;}
		if (dni > p.dni){ret= 1;}
		return ret;
	}
}
```

Main

```
Persona persona1 = new Persona("Juan");
Persona persona2 = new Persona("Pedro");
Agenda a = new Agenda<Persona>();
System.out.println(a. devolverLaPersonaMaschica(persona1,persona2));
```

Persona tiene que implementar la interfaz Comparable.
Esto se explicara mas adelante, en la clase de interfaces.
Los tipos predefinidos ya implementan dicha interfaz.