Docente: Juan Carlos Pérez P.
Alumno : _____**Fecha :** _____**Nota:**____

Justificación*:* Se pretende con éste contribuir a que el alumno reconozca las distintas conversiones datos en java.

Objetivos: Aplicar casting en la solución de un programa
Manejar algunas funciones básicas en la conversión de datos

Contenido.

Java Operador Cast y Conversiones

En java se puede forzar un dato, variable o una expresión a convertirse o cambiarse a un nuevo tipo de dato.
El operador cast realiza este proceso, es decir convierte datos, variables o expresiones a un nuevo tipo de dato, su formato es:
(nuevo tipo) dato , var, exp;
Ejemplo:

```
//declaración
int alfa;

// Asignación

alfa = 20;

// Cambio de tipo

(float) alfa;
```

Ejemplo:

```
(int) 3.1416;
```

-En este ejemplo se está convirtiendo un float a int, recordar que en este caso ya no se tendrán los decimales.
-Como nota importante se puede hacer Cast sólo con tipos de datos, no entre tipos de datos y objetos de una clase numérica.
Como ***nota importante*** este operador resuelve los dos problemas pendientes:
- El de la división entre enteros.
- El tipo de dato especifico que requieren las funciones.

Ejemplos:

a)**Declaración**
float alfa;

Operación

alfa = (float)23/5;

Pero en estos casos es preferible

alfa=23/5.0;

En toda división recordar agregar a uno de los dos valores el (.0), solo que los dos elementos sean variables entonces usar el operador cast con una de ellas.

double potencia;
potencia = Math.pow ((double)5, (double)3);

Como se observa en el ejemplo, se puede usar pow(), directamente con los datos, argumentos o parámetros requeridos , si estos son numéricos, pero transformándolos con el operador cast.
Esto también va a permitir evaluar expresiones matemáticas de manera más directa y sencilla, solo recordando usar un pow() por cada potencia y cada raíz de la ecuación, ejemplo: $y = 3x^3 - \sqrt[3]{x} + 4x^2$

dentro de un programa esto se resuelve con;

```
package operaciones;
import java.math.*;
import java.util.Scanner;
public class Potencia{
public  static void main(String args[])  {
double y, x;
Scanner leer = new Scanner(System.in);
System.out.print("Ingrese el valor de x");
x= leer.nextDouble();
y = 3 * Math.pow(x, (double)3) - Math.pow(x, (1/3.0)) + 4 * Math.pow(x, (double)2);
System.out.print("Resultado de y :"+ y + "Con x ="+ x );
  }
}
```

Nota : Para la captura de datos podemos también se puede utilizar:

```
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

En **java** jsp y **java** servlets para convertir tipos de datos numéricos a objetos numéricos y viceversa se deberán usar los métodos que traen la clase numéricas de manera apropiada, en general existen tres casos generales diferentes, considerando datos o variables numéricas, objetos numéricos y String que es un caso especial y común:

a.1) Variable numérica a variable numérica
Usar el operador cast ejemplo:
int alfa1=10; double alfa2=3.1416;
alfa1= (int)alfa2;
Observar que valores decimales deben declararse como double.

a.2) variable númerica a String
ejemplo;

int zeta=50;
String alfa= String.valueOf(zeta);
a.3) variable numérica a objeto numérico
caso 1: usando constructor
int zeta=80;
Float alfa4 = new Float(zeta);

caso 2: ya existe el objeto numérico
Double alfa5 = new Double(0);
Int zeta=30;
alfa5=Double.valueOf(String.valueOf(zeta));
b.1) Objeto Numérico a Variable numérica.
Todos los objetos numéricos deben crearse inicializados aunque sea a (0).
Integer alfa = new Integer(300);
Int zeta = alfa.intValue();
b.2) Objeto numérico a String
Float alfa = new Float("3.45");
String beta = new String(" ");
Beta = alfa.toString();
B.3) Objeto Numérico a Objeto Numérico
Integer alfa=new Integer(50);
Double beta=new Double(0);
beta = beta.valueOf(alfa.toString());
c.1) String to variable numérica
String alfa= new String("3.5");
double beta= 0;
beta= Double.parseDouble(alfa);
c.2) String to Objeto Numérico
String alfa8=new String("50");
Double alfa9=new Double(0);

alfa9 = alfa9.valueOf(alfa8);

Clases envoltorio (Wrapper)
Definición y uso de clases envoltorio

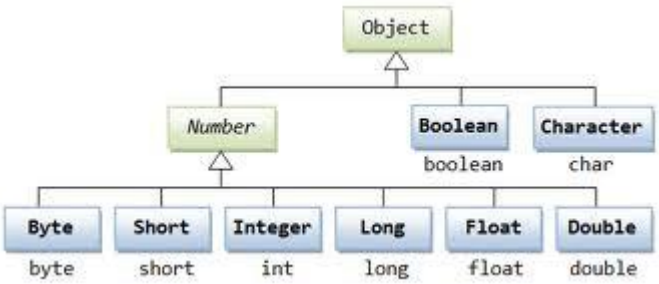
En ocasiones es muy conveniente poder tratar los datos primitivos (int, boolean, etc.) como objetos. Por ejemplo, los contenedores definidos por el API en el package java.util (Arrays dinámicos, listas enlazadas, colecciones, conjuntos, etc.) utilizan como unidad de almacenamiento la clase Object. Dado que Object es la raíz de toda la jerarquía de objetos en Java, estos contenedores pueden almacenar cualquier tipo de objetos. Pero los datos primitivos no son objetos, con lo que quedan en principio excluidos de estas posibilidades.

Para resolver esta situación el API de Java incorpora las clases envoltorio (wrapper class), que no son más que dotar a los datos primitivos con un envoltorio que permita tratarlos como objetos. Por ejemplo podríamos definir una clase envoltorio para los enteros, de forma bastante sencilla, con:

```
public class Entero {
   private int valor;

   Entero(int valor) {
      this.valor = valor;
   }

   int intValue() {
      return valor;
   }
}
```



La API de Java hace innecesario esta tarea al proporcionar un conjunto completo de clases envoltorio para todos los tipos primitivos. Adicionalmente a la funcionalidad básica que se muestra en el ejemplo las clases envoltorio proporcionan métodos de utilidad para la manipulación de datos primitivos (conversiones de / hacia datos primitivos, conversiones a String, etc.) Ver Fig. Wrapers
Las clases envoltorio existentes son:

- Byte para byte.
- Short para short.
- Integer para int.
- Long para long.
- Boolean para boolean
- Float para float.
- Double para double y
- Character para char.

Obsérvese que las clases envoltorio tienen siempre la primera letra en mayúsculas.

Las clases envoltura se usan como cualquier otra:

```
Integer i = new Integer(5);
int x = i.intValue();
```

Hay que tener en cuenta que las operaciones aritméticas habituales (suma, resta, multiplicación ...) están definidas solo para los datos primitivos por lo que las clases envoltura no sirven para este fin.

Las variables primitivas tienen mecanismos de reserva y liberación de memoria más eficaces y rápidos que los objetos por lo que deben usarse datos primitivos en lugar de sus correspondientes envolturas siempre que se pueda.

Método	**Descripción**
Integer(**int** valor) Integer(String valor)	Constructores a partir de int y String
int intValue() **byte** byteValue() **float** floatValue()	Devuelve el valor en distintos formatos, int, long, float, etc
boolean equals(Object obj)	Devuelve true si el objeto con el que se compara es un Integer y su valor es el mismo.
static Integer getInteger(String s)	Devuelve un Integer a partir de una cadena de caracteres. Estático
static int parseInt(String s)	Devuelve un int a partir de un String. Estático.
static String toBinaryString(**int** i) **static** String toOctalString(**int** i) **static** String toHexString(**int** i) **static** String toString(**int** i)	Convierte un entero a su representación en String en binario, octal, hexadecimal, etc. Estáticos
String toString()	
static Integer valueOf(String s)	Devuelve un Integer a partir de un String. Estático.

Actividades: Aplicar los conceptos vistos a algunos programas en java
Recursos: Software Netbeans, java 2, Guía de aprendizaje
http://www.juanperezu.com

Bibliografía: http://www.programacionfacil.com/java:operador_cast
www.java.sum.com,
www.lawebdelprogramador.com
Piensa en Java ,Mac Graw Hill.