Object Oriented Programming Lab

CSE 1206

LAB 4

Course Teacher

Dr. Shahriar Mahbub, Professor
Nibir Chandra Mandal, Lectuer
Nowshin Nawar Arony, Lecturer

Type Casting

- **When two types are incompatible and destination type is smaller than source type, we do type casting to convert them.**

(target-type) value

Example:

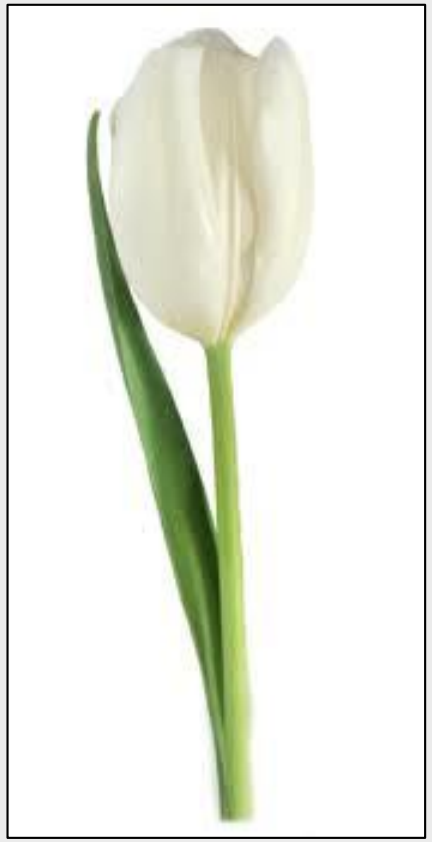
```
double dbNum = 5.8;
int num = (int)dbNum;
```

RED Tulip

Object 1

+

White Tulip



Object 2

=

Pink Tulip

New Object

1. Project Name: **CreateTulip**

2. Another Class name: **Tulip**

3. Variables in Tulip class:
   - private String color;
   - private double height;

4. Declare a default **empty constructor**, **parameterized constructor** and respective **getter setter methods in Tulip class.**

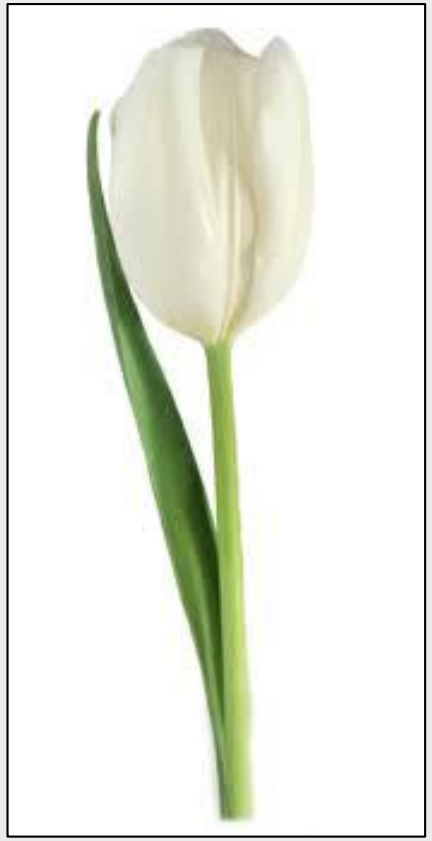
5. Create two objects of **Tulip** in **CreateTulip** and initialize the variables using the parameterized constructor.

```java
public class CreateTulip {

    public static void main(String[] args) {

        Tulip redTulip= new Tulip("red",2);
        Tulip whiteTulip = new Tulip("white",4);

        System.out.println("Properties of 1st Tulip object");
        System.out.println(redTulip.getHeight() + " feet");
        System.out.println(redTulip.getColor());

         System.out.println("\nProperties of 2nd Tulip object");
        System.out.println(whiteTulip.getHeight() + " feet");
        System.out.println(whiteTulip.getColor());
```

RED Tulip

Object 1

White Tulip



Object 2

Objects As Return Types and Parameters in Methods

- Create this method inside the class **Tulip**

```
public Tulip mixTulip(Tulip anotherTulip)
{
    Tulip newTulip = new Tulip();

    newTulip.height = (this.height + anotherTulip.height)/2;

    return newTulip;
}
```

- Call the method in the **CreateTulip** class

```
Tulip mixedTulip1=redTulip.mixTulip(whiteTulip);

System.out.println("\nProperties of Mixed Tulip object");
System.out.println(mixedTulip1.getHeight());
```

- Lets complete the method by changing the color also.

```
public Tulip mixTulip(Tulip anotherTulip) {
    Tulip newTulip = new Tulip();

    newTulip.height = (this.height + anotherTulip.height) / 2;

    if (this.color == "red" && anotherTulip.color == "white")
    {
        newTulip.color = "pink";
    }
    return newTulip;
}
```

- Print the color in the **CreateTulip** class

```
Tulip mixedTulip1 = redTulip.mixTulip(whiteTulip);

System.out.println("\nProperties of Mixed Tulip object");
System.out.println(mixedTulip1.getHeight());
System.out.println(mixedTulip1.getColor());
```

Method Overloading

1. Java allows different methods to have same name.
2. Conditions:
   - By changing **number of arguments.**
   - By changing the **data type of arguments.**

Now Create a new Project Named MethodOverloading.

Inside this Project Create a new Class Named Adder.

```java
package methodoverloading;

public class Adder {

    public void add(int a, int b) {
        System.out.println(a+b);

    }
    public void add(int a, int b, int c) {
        System.out.println(a+b+c);
    }
```

Two methods having same name add but number of arguments in the parameter are different

```java
package methodoverloading;

public class MethodOverloading {

    public static void main(String[] args) {

        Adder objAdder=new Adder();
        objAdder.add(2, 3);
        objAdder.add(2, 3, 4);
    }

}
```

```java
package methodoverloading;

public class Adder {

    public void add(int a, int b) {
        System.out.println(a+b);
    }
    public void add(double a, double b) {
        System.out.println(a+b);
    }
    public void add(int a, int b, int c) {
        System.out.println(a+b+c);
    }
    public void add(double a, double b, double c) {
        System.out.println(a+b+c);
    }
}
```

Methods having same name add and same number of arguments but type is different

```java
package methodoverloading;

public class MethodOverloading {

    public static void main(String[] args) {

        Adder objAdder=new Adder();
        objAdder.add(2, 3);
        objAdder.add(2, 3, 4);
        objAdder.add(5.6, 1.2);
        objAdder.add(2.6, 3.3, 4.5);
    }

}
```

Changing the **return type** is not counted as method overloading

Going back to Tulip class

1. What if we want to mix 3 different colored tulips?
2. Then we can pass two objects of Tulip class in the parameter.
3. For that lets create another method.

```java
public Tulip mixTulip(Tulip tulip2, Tulip tulip3) {
    Tulip newTulip = new Tulip();

    newTulip.height = (this.height + tulip2.height
            + tulip3.height) / 3;

    newTulip.color = this.color + " " + tulip2.color
            + " " + tulip3.color;

    return newTulip;
}
```

```java
public class CreateTulip {

    public static void main(String[] args) {

        Tulip redTulip = new Tulip("red", 2);
        Tulip whiteTulip = new Tulip("white", 4);
        Tulip yellowTulip = new Tulip("yellow", 5);

        Tulip mixedTulip2 =
                redTulip.mixTulip(yellowTulip, whiteTulip);

        System.out.println(mixedTulip2.getHeight());
        System.out.println(mixedTulip2.getColor());
```

Type Promotion Rules

- **When two types are compatible and destination type is larger than source type.**
  - **byte -> short || int || long || float || double**
  - **short -> int || long || float || double**
  - **char -> int || long || float || double**
  - **int -> long || float ||double**
  - **long -> float || double**
  - **float -> double**

***Read Type Conversion and Casting in Chapter 3 from the book Java The Complete Reference by Herbert Schildt.**

(Page 48 to 51 in eight edition)

```java
package methodoverloading;

public class Adder {

    public void add(double a, double b) {
        System.out.println(a+b);
    }

    public void add(double a, double b, double c) {
        System.out.println(a+b+c);
    }
}
```

```java
package methodoverloading;

public class MethodOverloading {

    public static void main(String[] args) {

        Adder objAdder=new Adder();
        int a=2, b=3, c=4;

        objAdder.add(a, b);
        objAdder.add(a, b, c);
        objAdder.add(5.6, 1.2);
        objAdder.add(2.6, 3.3, 4.5);
    }

}
```

Although Integers but Java automatically converts them to double