CIS 351-Data Structure-Method Overloading Jan 28, 2020

Dr. Farzana Rahman

Syracuse University

Quiz 1: What Will Be Printed?

```
public class MethodDemo {

   public static void main(String[] args) {
       int a;
       int b;
       int c;

       a = 2;
       b = 3;
       c = methodOne(b, a);
       System.out.println(a + " " + b + " " + c);
   }

   public static int methodOne(int a, int b) {
      int result;

      result = a * 2 + b;
      a = 6;
      return result;
   }

}
```

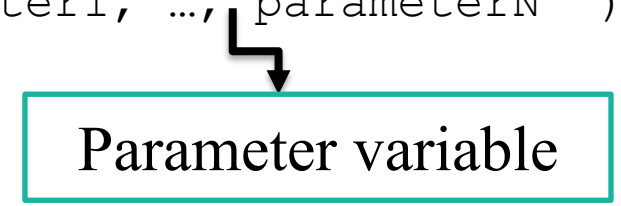
Passing argument

- A method can be written to accept argument:

```
System.out.println(str);
```

- General format of method declaration:

```
return-value-type  method-name(  parameter1, …, parameterN  )
{
      declarations and statements
}
```



- Multiple argument passing is allowed in java
- How to pass argument:

```
String str = "Hello";
System.out.println(str);
```

- Be sure that argument's data type is compatible with parameter variable type.
- Java does widening conversion if argument type ranked lower than parameter type.

Two Types of Parameter Passing

- In java, all argument of primitive data types are passed by value.
  - In this case, modification of the *formal argument* has no effect on the *actual argument*, - it is call by value
- If we pass any object as an argument, the memory address held by that object variable is passed
  - In this case, modification of the *formal argument* can change the value of the *actual argument*, - it is call by reference
  - The pointer to the object is passed, not the object itself
- However, String object is immutable in java- they do not change.

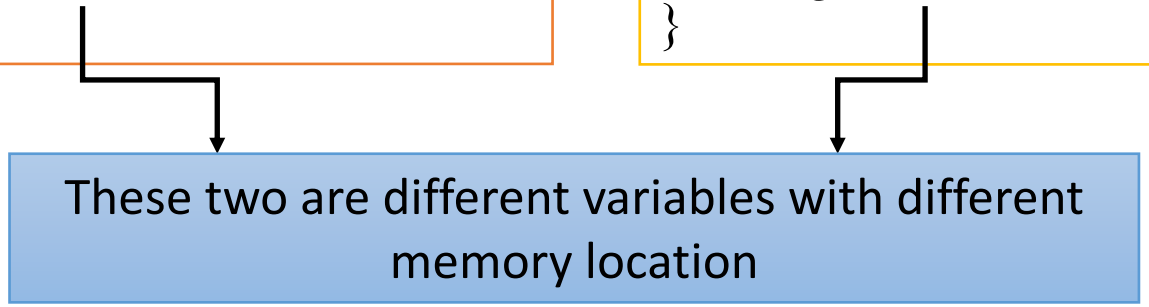
Scoping Rule

- A local variable is declared inside a method and it is not accessible to statements outside the method

- Different methods can have local variables of the same name, since methods cannot see each other's local variable

```
void showName1()
{
    string name = "Bob";
}
```

```
void showName2()
{
    string name = "Alice";
}
```



- A method's local variable exists only during the execution of the method. This is known as the lifetime of the *variable*.

Returning value from a method

- A method may send a value back to the statements that called the method – are called value returning method

```
int num;
num = keyboard.nextInt();
```

- Defining a value retuning method

```
public int  calculateMin(int num1, int num2)
{   ……
    int minimum_value= /*calculate the minimum*/
    return minimum_value;
}
```

- It can be either a primitive data type variable
- It can be an object too

- How to call this method

```
int n1 = 10;
int n2 = 100;
int MinValue = calculateMin(n1, n2);
```

Method Overloading in JAVA

- It turns out that Java allows a programmer to write two or more methods with the same name in the same class
- This is called method overloading
- The only requirement is that each method declaration has to be different from the others by the parameter type, position, or number

Method Overloading (cont.)

- When the Java compiler tries to decipher overloaded methods it looks at the method signature
  - which is the combination of the method name and the number and types of its parameters

Example (method overloading)

```
Public void calcArea(int x, int y)
{

}
```

```
Public void calcArea(double r)
{

}
```

```
Public void calcArea(int x, int y, double r)
{

}
```

Example (method overloading)

Circle c = new Circle();

c.calArea(3,3); →

```
Public void calcArea(int x, int y)
{

}
```

c.calArea(5.3); →

```
Public void calcArea(double r)
{

}
```

c.calArea(3, 3, 5.3); →

```
Public void calcArea(int x, int y, double r)
{

}
```

The Keyword this

- Use this to refer to the current object.
- Use this to invoke other constructors of the object.