Lecture (08)
Methods

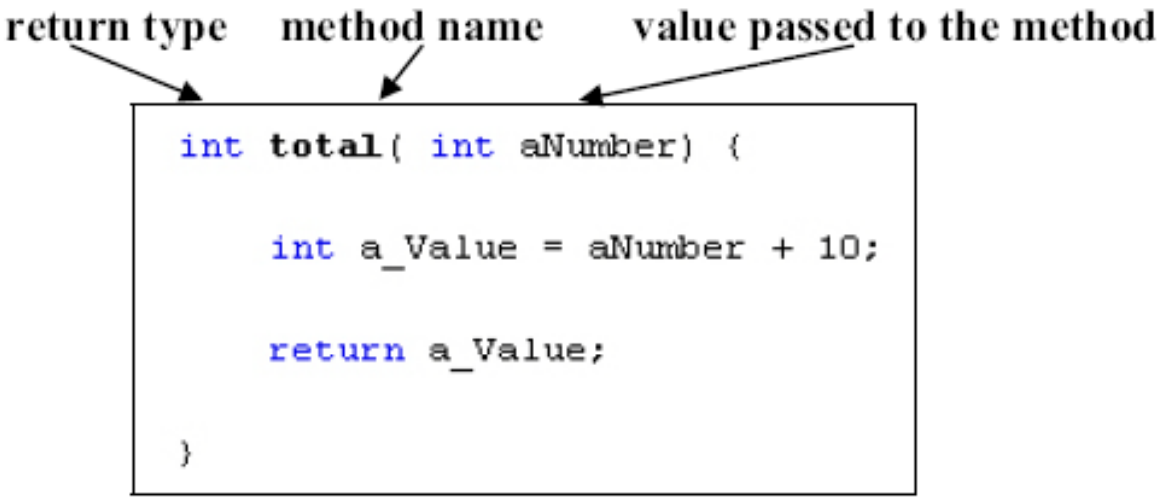
Dr. Ahmed ElShafee

Agenda

- Introduction
- Build & call your methods
- Method types (return value)
- Method types (return nothing)
- Method types (with arguments)
- Passing more than one value to your methods

Introduction

- A method is just a chunk of code that does a particular job.
- But methods are set out in a certain way.
- You have a method header, and a method body.
- The header is where you tell Java what value type, if any, the method will return (an int value, a double value, a string value, etc).
- As well as the return type, you need a name for your method, which also goes in the header.
- You can pass values over to your methods, and these go between a pair of round brackets.
- The method body is where you code goes.



- the value that you want to return from your method, after your code has been executed, written after **return** key word, and should be in the same data type of method header

- Sometimes you don’t want Java to return anything at all.
- A method that doesn’t return any value at all can be set up with the word void.

```
void print_text(String someText) {

    System.out.println( "Some Text Here" );

}
```

- Methods don’t need to have values passed to them.
- You can just execute some code.

```
void print_text() {

    System.out.println( "Some Text Here" );

}
```

- And here's an **int** method that has no values being passed

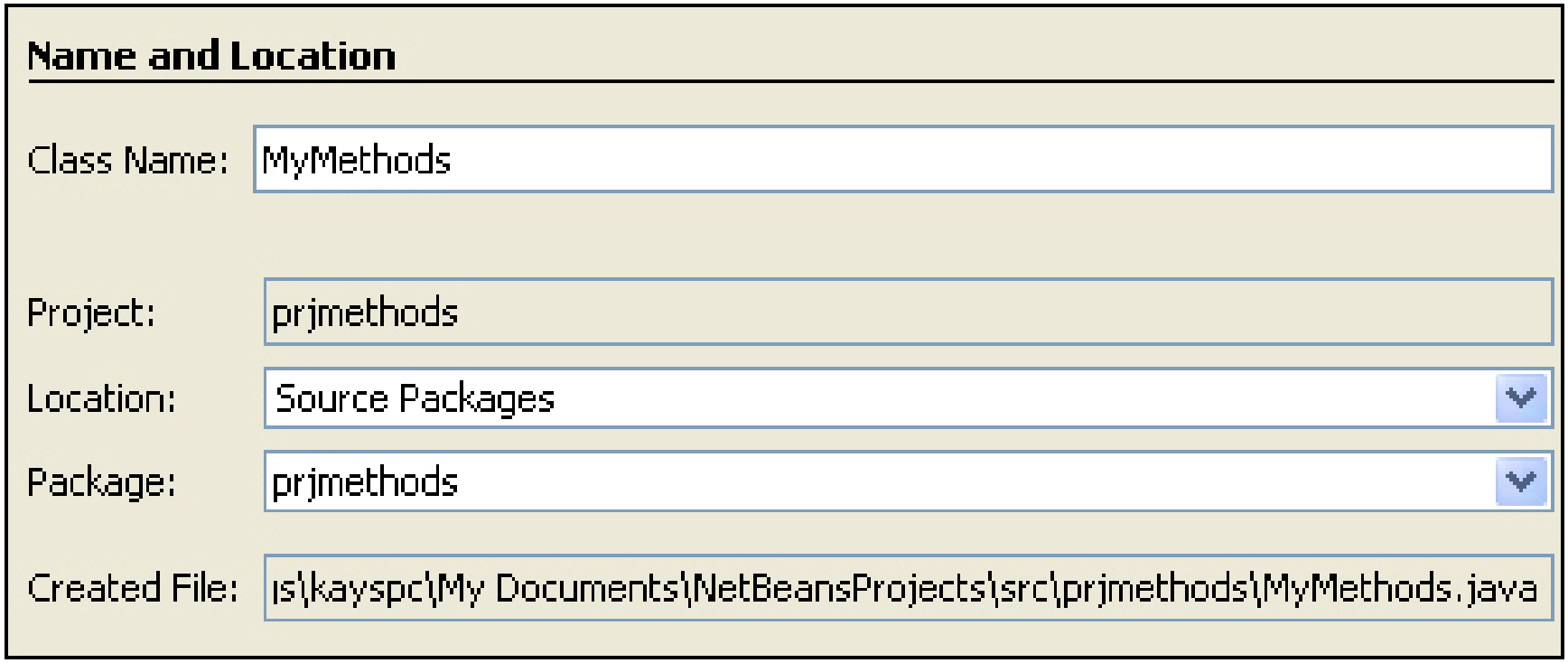
```
int total() {

    int a_Value = 10 + 10;

    return a_Value;

}
```

Build & call your methods

- Start a new Java Application project.
- Give your project a name, and rename the
- Main method to something else.
- Then click **Finish.**
- **we've** called our project **prjmethods, and the class TestMethods:**

```
package prjmethods;

public class TestMethods {

    public static void main(String[] args) {


    }

}
```

- add a new class to your project, click **File from the NetBeans menu.**
- From the **File** menu, select **New File.**
- You'll see a dialogue box appear.
- In the Categori**es** section select **Java,** and in the File Types section select **Java Class.**
- Then click the **Next button at the bottom.**
- **In step two, type a name for your new class.**
- We've called ours **MyMethods.**
- You can leave everything else on the defaults:


Name and Location
Class Name:
MyMethods
Project:
prjmethods
Location:
Source Packages
Package:
prjmethods
Created File:
|s\kayspc\My Documents\NetBeansProjects\src\prjmethods\MyMethods.java

- A new tab will appear in the NetBeans software, with some default comments about how to change templates.

```
package prjmethods;


public class MyMethods {


}
```

- The thing to notice is that there's no Main method this time – just a blank class with the name you chose, and a pair of curly brackets for your code.
- Let's add one of our methods.

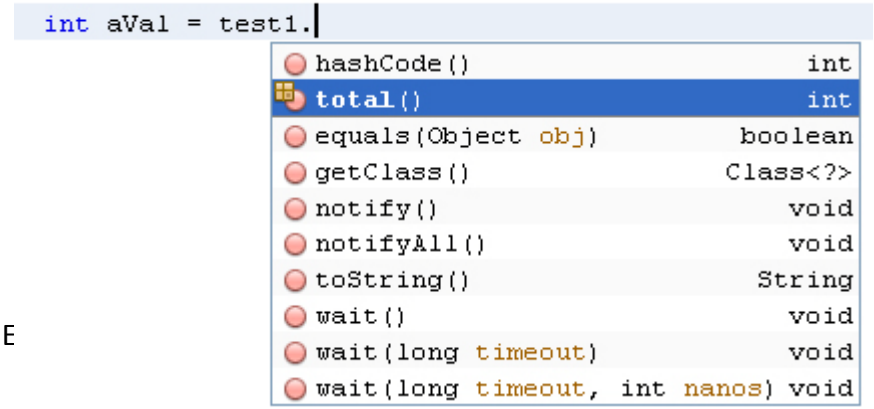
Method types (return value)

```
package prjmethods;

public class MyMethods {

      int total() {
         int a_Value = 10 + 10;

         return a_Value;
      }

}
```

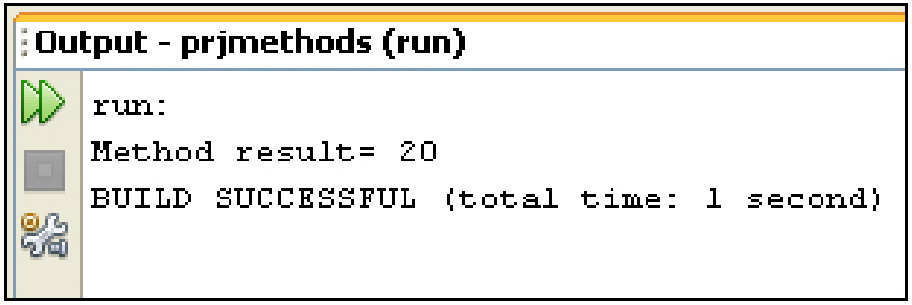
- To call the total method, select your **TestMethods tab in NetBeans,** the one with your Main method.
- We're going to call the **total** method from the **Main** method**.**

```
public class TestMethods {

    public static void main(String[] args) {

        MyMethods test1 = new MyMethods();

        int aVal = test1.total();

        System.out.println( "Method result= " + aVal );

    }
}
```

When you type object name, NetBeans will list all its associated methods and properties



- Run your code



- So to call a method that returns a value, note what value is being returned by your method.
- Then assign this value to a new variable, **aVal** in our case.

Method types (return nothing)

- If your method is of type void, however, you don't need to assign it to a new variable like **aVal.**
- **As an example,**

```
package prjmethods;

public class MyMethods {

     int total() {
        int a_Value = 10 + 10;

        return a_Value;
    }

     void print_text() {

        System.out.println( "Some Text Here" );
    }

}
```

- it has an empty pair of round brackets as we're not handing it any values.

- Call method from main function

```
System.out.println( "Method result= " + aVal );

test1.
```

```
equals(Object obj)              boolean
getClass()                      Class<?>
hashCode()                      int
notify()                        void
notifyAll()                     void
print_text()                    void
toString()                      String
total()                         int
wait()                          void
wait(long timeout)              void
wait(long timeout, int nanos)   void
```

- run

```
Output - prjmethods (run)
run:
Method result= 20
Some Text Here
BUILD SUCCESSFUL (total time: 2 seconds)
```

Method types (with arguments)

- You can pass values to your methods so that something can be done with them.
- The values go between the round brackets of the method.
- Example:

```
package prjmethods;

public class MyMethods {

     int total() {
        int a_Value = 10 + 10;

        return a_Value;
     }

     void print_text() {

        System.out.println( "Some Text Here" );
     }

     int total(int aNumber) {
        int a_Value = aNumber + 20;

        return a_Value;
     }
}
```

- We now have two methods with the same name: **total.**
- The difference between the two is that this new one has a value between the round brackets.
- Java allows you to do this, and it is called **method overloading.**

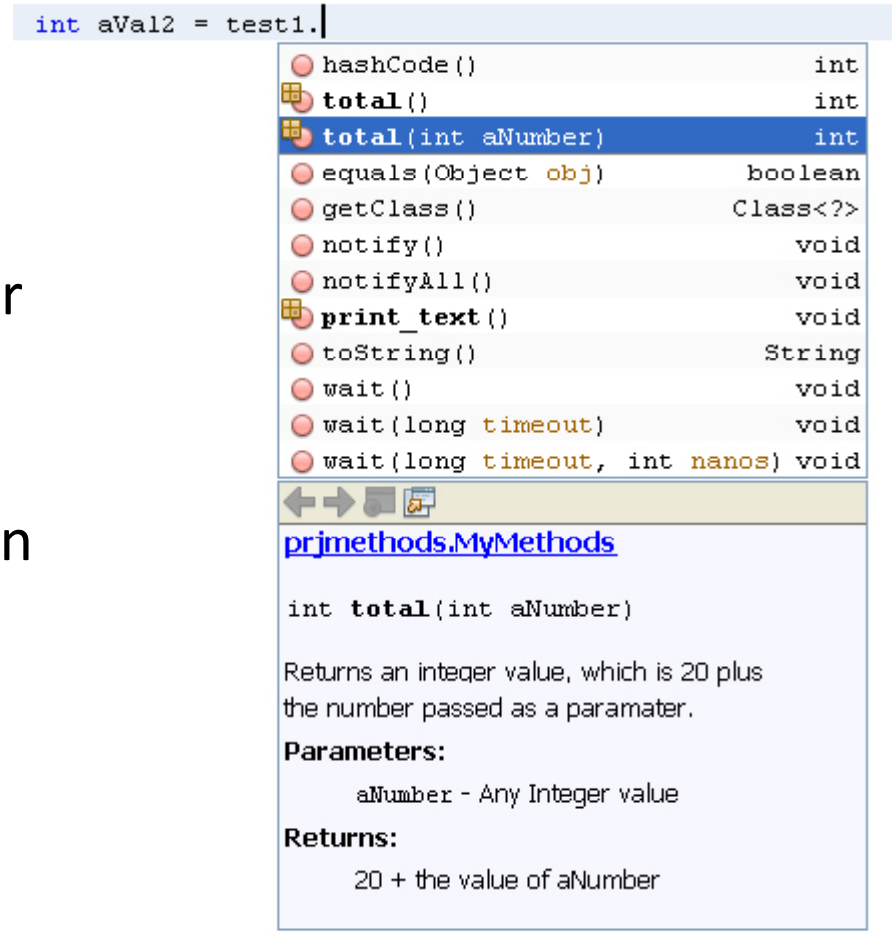
- Add javaDoc comments
- Before you try out your new method, add some comments directly above the method:

```
/**
* Returns an integer value, which is
* 20 plus the number passed as a paramater.
*       @param aNumber  Any Integer value
*       @return 20 + the value of aNumber
*/
int total(int aNumber) {
   int a_Value = aNumber + 20;

   return a_Value;
}
```

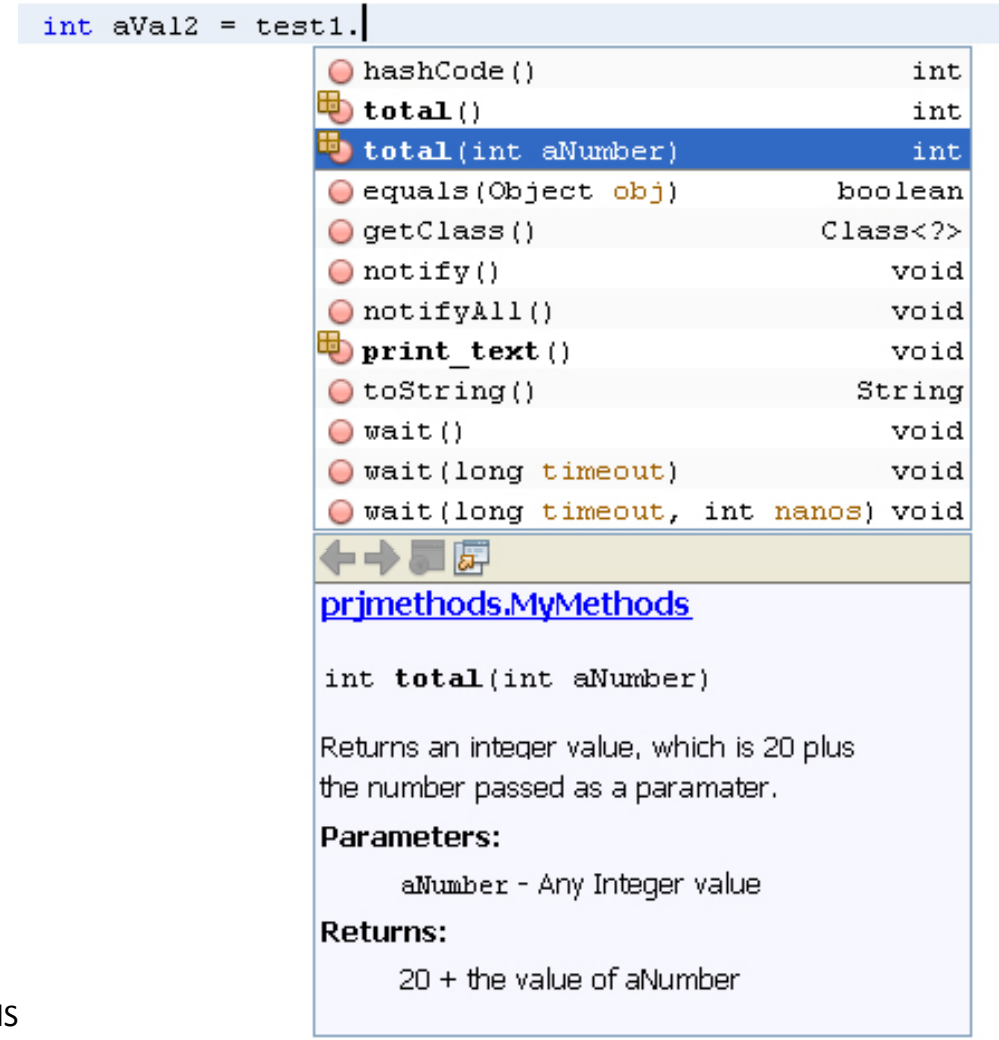
- Now switch back to your code and add the following line:

  int aVal2 = test1.total(30);

- As soon as you type the dot after your **test1** object, you'll see the popup list again.
- Your new **total** method will be on it.
- Click on the new method to highlight it and NetBeans will display the following:



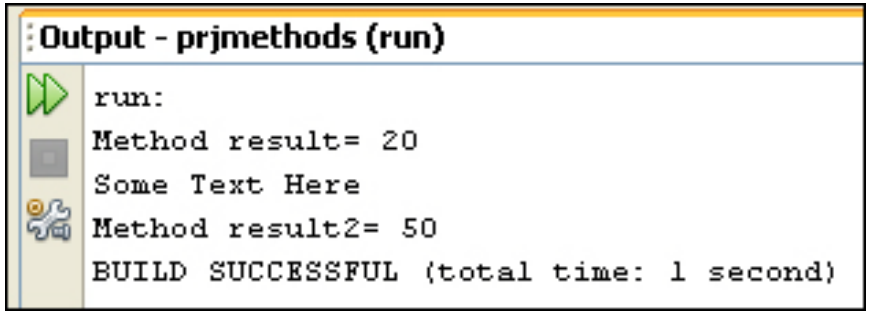
- The comments that we added are now displayed in the blue box underneath the list of methods.



- Code

```
public class testMethods {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic heremyMethods test1=new myMethods();
        myMethods test1=new myMethods();
        int aVal=test1.total();
        System.out.println("Method Result= "+aVal);
        test1.print_text();
        aVal=test1.total(30);

        System.out.println("Method Result= "+aVal);


    }
}
```

- run

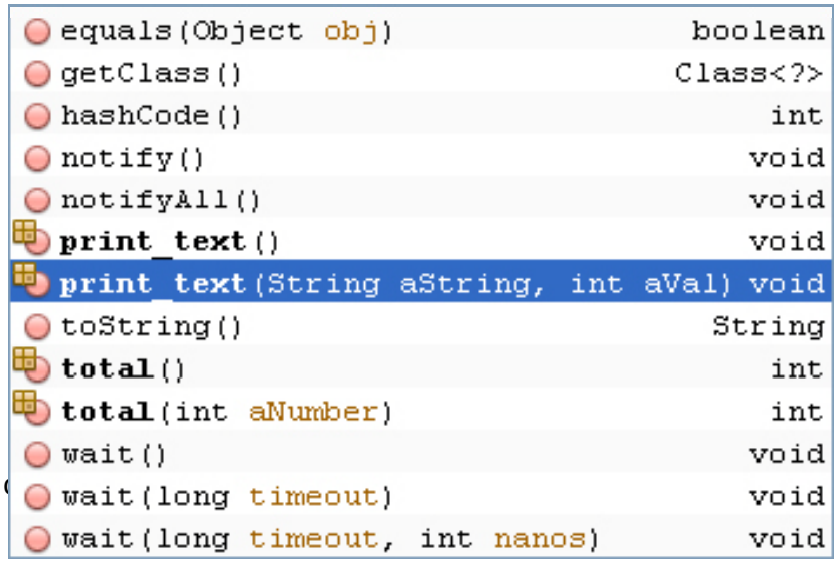


Passing more than one value to your methods

- You can pass more than one value over to your methods.
- Add the following method to your **MyMethods class:**

```
void print_text(String aString, int aVal) {

    System.out.println( aString  + aVal );

}
```

- Go back to your TestMethods class and make the following call to the method:



- Code

```
public static void main(String[] args) {

    MyMethods test1 = new MyMethods();

    int aVal = test1.total();

    System.out.println( "Method result= " + aVal );

    test1.print_text();

    int aVal2 = test1.total(30);
    System.out.println( "Method result2= " + aVal2 );

    test1.print_text("The value was ", aVal2);

}
```

- run

```
Output - prjmethods (run)
run:
Method result= 20
Some Text Here
Method result2= 50
The value was 50
BUILD SUCCESSFUL (total time: 0 seconds)
```

Thanks,

See you next Lecture, isA