Advanced Java Programming

Part 5: The `this` reference and the `this()` method. Useful methods to define for your classes: `toString()`, `equals()`

James Tam

Self Reference: The 'This' Reference

- **New term, this reference**: A reference to the object (called the "`this`" reference)
- From every (non-static) method of an object there exists a reference to the object (called the "`this`" reference) [1]

```
main(String args []) {
  int x;
  Person fred = new Person();
  Person barney = new Person();
  fred.setAge(35);
}

public class Person {
    private int age;
    public void setAge(int anAge) {
          age = anAge;
   }
            ...
 }
```

This is one reason why methods must be invoked via a reference name (the contents of the reference 'fred' will be copied into the '`this`' reference (both point to the 'Fred' object).

The '`this`' reference is implicitly passed as a parameter to all non-static methods. One use of '`this`' is to distinguish which object's method is being invoked (in this case Fred vs. Barney)

1 Similar to the '`self`' keyword of Python except that '`this`' is a syntactically enforced name (can't use another name).

James Tam

The 'This' Reference Is Automatically Referenced Inside (Non-Static) Methods

```
public class Person {
   private int age;
    public void setAge(int anAge) {
        // These two statements are equivalent
        age = anAge;
        this.age = anAge;
   }
}
```

James Tam

Parameter Types: Explicit Vs. Implicit

- **New term, explicit parameter(s)**: explicitly passed (you can see them in the round brackets when the method is called and defined).
  - Method calls
    ```
    fred.setAge(10);    //10 explicit
    barney.setAge(num);   //num explicit
    ```
  - Method definition
    ```
    public void setAge(int age) { ... }  //age explicit
    ```
- **New term, implicit parameter**: implicitly passed into a method (automatically passed and cannot be explicitly passed): the '**this**' reference.
  - Method call
    ```
    Person bill = new Person();
    bill.setAge(100);
    ```
  - Method definition
    ```
    public void setAge(int age) {this.age = age;} //'this' is implicit
    ```

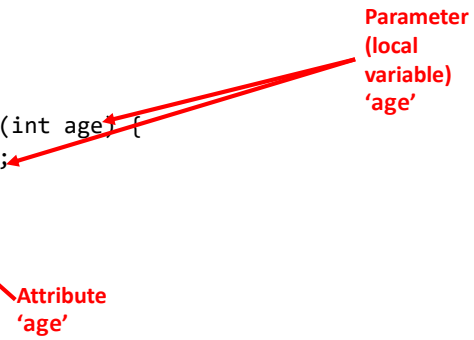
James Tam

Benefits Of 'This': Attributes

- One side benefit is the `this` reference can make it very clear which attributes are being accessed/modified.

```
public class Person
{
    private int age;

    public void setAge(int age) {
        this.age = age;
    }
}
```



James Tam

Benefits Of 'This': Parameters

- Another side benefit is the `this` reference can make it clear which object is being accessed e.g., when a class method takes as a explicit parameter an instance of that class[1]

```
main (String [] args) {
    Person fred = new Person("Fred");
    Person barney = new Person("Barney");
    barney.nameBestBuddy(fred);    // JT: Explicit? Implicit?
}
// JT: What will be the output?
public void nameBestBuddy(Person aPerson) {
    println(this.name + " best friend is " + aPerson.name);
}
```

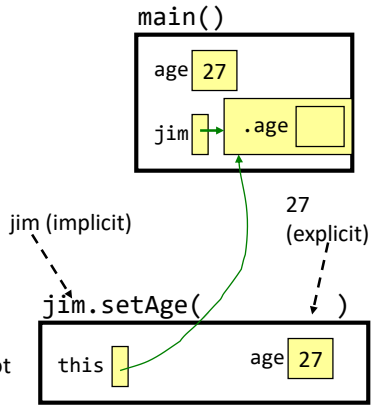
1 JT: more on class methods that take parameters which are of the class type (e.g. Person) later – see the 'equals()' method

James Tam

Main Benefit Of 'This': Scope

- Recall: according to scoping rules, local variables are not accessible outside of that function or method (unless returned back to the caller or passed into another method).
  - The `this` implicit parameter can allow access to these locals.

```
main (String [] args) {
    int age = 27;
    Person jim = new Person();
    jim.setAge(age);
}
class Person {
    public void setAge(int age) {
        this.age = age;
}
```



Normally the object referred to by the '`jim`' reference not accessible outside of `main()` but the '`this`' reference contains it's address (implicit pass by reference)

James Tam

Static Methods: No 'This' Reference

- Recall: **static methods** do not require an object to be instantiated because they are invoked via the **class name** not a reference name.

```
int result = Math.abs(-12);
```

- That means static methods do not have the implicit 'this' parameter passed in.
- Also recall I said for now avoid [for the '`Driver`' class]:
  - Defining attributes for the Driver
  - Defining methods for the Driver (other than the main method)
  - Unless the attributes and the methods are static they cannot be accessed without instantiating the Driver class.

James Tam

This()

- Can be used when constructors have been overloaded.
- Calls one version of the constructor from another constructor.
- **Name of the folder containing the complete example** : `11thisMethod`

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Person aPerson = new Person();
        aPerson.show();

        aPerson = new Person(99);
        aPerson.show();

        aPerson = new Person("Bob");
        aPerson.show();
    }
}
```

James Tam

Class Person

```
public class Person {
    private int age;
    private String name;

    public Person() {
        age = -1;
        name = "none";
    }

    public Person(int anAge) {
        this();
        age = anAge;
    }
```

James Tam

Class Person (2)

```
    public Person(String aName) {
        this();
        name = aName;
    }

    public void show()
    {
        System.out.println(age + " " + name);
    }
}
```

James Tam

Displaying State: Evaluating The Previous Program

```
public void show()
{
    System.out.println(age + " " + name);
}
```

James Tam

Displaying The Current State Of Objects

- The `toString()` method provides information about the state of a particular object (contents of important attributes).
  - Returns a string representation of the state (current value of variable attributes and any other relevant information).
- It's automatically be called whenever a reference to an object is passed as a parameter to "`print()/println()`".

```
E.g. class Person { … }

…
Person p = new Person();
//The following instructions are equivalent
…println(p);
…println(p.toString());
```

James Tam

toString() Example

- **Name of the folder containing the complete example** :
  12toString

Class Person

```
public class Person
{
    private int height;
    private int weight;
    private String name;

    public Person(String name, int height, int weight)
    {
        this.name = name;
        this.height = height;
        this.weight = weight;
    }
```

Class Person (2)

```
    public String getName()
    {
        return(name);
    }

    public int getHeight()
    {
        return(height);
    }

    public int getWeight()
    {
        return(weight);
    }
```

James Tam

Class Person (3)

```
    public String toString()
    {
        String s;
        s = "Name: " + name + "\t";
        s = s + "Height: " + height + "\t";
        s = s + "Weight: " + weight + "\t";
        return(s);
    }
}
```

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Person jim = new Person("Jim",69,160);
        System.out.println("Attributes via accessors()");
        System.out.println("\t" + jim.getName() + " " +
                           jim.getHeight() +
                          " " + jim.getWeight());
```

```
Attributes via accessors()
        Jim 69 160
```

```
        System.out.println("Attributes via toString()");
        System.out.println(jim);
```

```
Attributes via toString()
Name: Jim       Height: 69      Weight: 160
```

```
    }
}
```

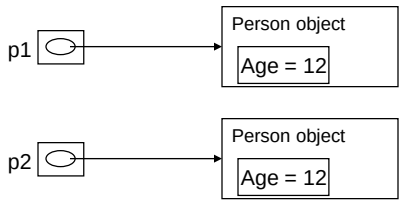
James Tam

Comparing Objects

- Recall from the discussion of parameter passing (pass by reference) that a reference contains the address of an object or array.
- Using the comparison operator on the references '==' will only determine if the address (and not data) is the same.

```
Person p1 = new Person(12);
Person p2 = new Person(12);


if (p1 == p2)
```



James Tam

Comparing Objects (2)

- Either each attribute of each object must be manually compared or else some form of `equals()` method must be implemented.

James Tam

Implementing `Equals()`

- **Name of the folder containing the complete example**:
  `13equals`

James Tam

Class Person

```
public class Person {
    private int height;
    private int weight;

    public Person(int height, int weight) {
        this.height = height;
        this.weight = weight;
    }

    public int getHeight() {
        return(height);
    }

    public int getWeight() {
        return(weight);
    }
```

James Tam

Class Person (2)

```
    public void setHeight(int height) {
        this.height = height;
    }


    public void setWeight(int weight) {
        this.weight = weight;
    }
```

Implicit: **Jim** Explicit: **Bob**

```
    public boolean equals(Person compareTo) {
        boolean flag = true;
        //Access to compareTo's private attributes allowed here!
        if (this.height != compareTo.height ||
            this.weight != compareTo.weight)
            flag = false;
        return(flag);
    }
}
```

James Tam

The Driver Class

```
public class Driver
{
    public static void main(String [] args)
    {
        Person jim = new Person(69,160);
        Person bob = new Person(72,175);
```

James Tam

The Driver Class (2)

```
new
Person(69,160);
new
Person(72,175);
```

```
        System.out.println("Different data, addresses");
        System.out.println("Compare data via accessors()");
        if (jim.getHeight() == bob.getHeight() &&
            jim.getWeight() == bob.getWeight())
            System.out.println("\tObjects same data");
        else
            System.out.println("\tNot equal");

        System.out.println("Compare data via equals()");
        if (jim.equals(bob) == true)
            System.out.println("\tObjects same data");
        else
            System.out.println("\tNot equal");

        System.out.println("Compare addresses");
        if (jim == bob)
            System.out.println("\tSame address");
        else
            System.out.println("\tDifferent addresses");
```

```
Compare data via accessors()
        Not equal
```

```
Compare data via equals()
        Not equal
```

```
Compare addresses
        Different addresses
```

James Tam

Person(72,175); # via set()

Person(72,175);

The Driver Class (3)

```
System.out.println();
System.out.println("Same data, different addresses");
jim.setHeight(72);
jim.setWeight(175);
if (jim.equals(bob) == true)
    System.out.println("\tObjects same data");
else
    System.out.println("\tNot equal");


System.out.println("Compare addresses");
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");
```

```
Same data, different addresses
        Objects same data
Compare addresses
        Different addresses
```

James Tam

Person(72,175); # via set()

Person(72,175);

The Driver Class (4)

```
System.out.println();
System.out.println("Same data, different addresses");
jim.setHeight(72);
jim.setWeight(175);
if (jim.equals(bob) == true)
    System.out.println("\tObjects same data");
else
    System.out.println("\tNot equal");


System.out.println("Compare addresses");
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");
```

```
Same data, different addresses
        Objects same data
Compare addresses
        Different addresses
```

James Tam

jim = bob;

The Driver Class (5)

```
System.out.println();
System.out.println("Same addresses");
jim = bob;
if (jim == bob)
    System.out.println("\tSame address");
else
    System.out.println("\tDifferent addresses");
```

```
Same  addresses
        Same address
```

James Tam

After This Section You Should Now Know

- What is the `this` reference, why is it needed, how does it work
- What is the `this()` method and how does it work
- Why and how to define these methods for your class definitions
  - `toString`
  - `equals`

James Tam

Copyright Notification

- “Unless otherwise indicated, all images in this presentation are used with permission from Microsoft.”