Method Overloading, the `toString()`, and the `equals()` methods

Shirley B. Chu

July 22, 2020

De La Salle University
College of Computer Studies

Recall: Overloading Constructors

Overloading constructors allows us to have multiple ways of initializing an instance.

These constructors have different constructor signatures.

Method Overloading

- Methods can also be overloaded.

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data
- Rules:

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data
- Rules:
  - same method name

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data
- Rules:
  - same method name
  - different parameters

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data
- Rules:
  - same method name
  - different parameters
    - number of parameters

```
public void display ();
public void display (String name);
```

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data
- Rules:
  - same method name
  - different parameters
    - number of parameters

```
public void display ();
public void display (String name);
```

    - type of parameters

```
public void display (int age);
public void display (String name);
```

Method Overloading

- Methods can also be overloaded.
- Performs the same activity or operation, given different data
- Rules:
  - same method name
  - different parameters
    - number of parameters

```
public void display ();
public void display (String name);
```

    - type of parameters

```
public void display (int age);
public void display (String name);
```

    - order of parameters

```
public void display (int age, String school);
public void display (String name, int age);
```

Activity: `MyDate` class

Modify your `MyDate` class to allow updating of date value by providing the new month, day, and year. The date value can also be updated by changing the month value, either by providing the numerical value of the new month or the name of the new month.

Reference Type

- Arrays, and all defined classes are reference types.

Reference Type

- Arrays, and all defined classes are reference types.
- All reference types are also of type `Object`, which has the following methods:

Reference Type

- Arrays, and all defined classes are reference types.
- All reference types are also of type `Object`, which has the following methods:
  - `toString()`

Reference Type

- Arrays, and all defined classes are reference types.
- All reference types are also of type `Object`, which has the following methods:
  - `toString()`
  - `equals()`

`toString()`

- signature: `public String toString ()`

`toString()`

- signature: `public String toString ()`
- returns the *textual representation* of the object

`toString()`

- signature: `public String toString ()`
- returns the *textual representation* of the object
- recommended that all subclasses override this method

toString()

- signature: `public String toString ()`
- returns the *textual representation* of the object
- recommended that all subclasses override this method

```
public class MyDate
{   private int month, day, year; /*not a good coding practice */
                ⋮
}
```

toString()

- signature: `public String toString ()`
- returns the *textual representation* of the object
- recommended that all subclasses override this method

```
public class MyDate
{   private int month, day, year; /*not a good coding practice */
                ⋮
    @Override
    public String toString ()
    {
        return month + "/" + day + "/" + year;
    }
}
```

toString()

- signature: `public String toString ()`
- returns the *textual representation* of the object
- recommended that all subclasses override this method

```
public class MyDate
{   private int month, day, year; /*not a good coding practice */
                ⋮
    @Override
    public String toString ()
    {
        return month + "/" + day + "/" + year;
    }
}
```

`equals()`

- signature: `public boolean equals (Object obj)`

`equals()`

- signature: `public boolean equals (Object obj)`
- tells whether some other object `obj` is *equal* to this instance.

`equals()`

- signature: `public boolean equals (Object obj)`
- tells whether some other object `obj` is *equal* to this instance.
- by default, this method uses the `==` operator.

`equals()`

- signature: `public boolean equals (Object obj)`
- tells whether some other object `obj` is *equal* to this instance.
- by default, this method uses the `==` operator.
- has to be overriden to be useful.

equals()

- signature: `public boolean equals (Object obj)`
- tells whether some other object `obj` is *equal* to this instance.
- by default, this method uses the `==` operator.
- has to be overriden to be useful.

```
public class MyDate
{   private int month, day, year; /*not a good coding practice */
                ⋮
}
```

equals()

- signature: `public boolean equals (Object obj)`
- tells whether some other object `obj` is *equal* to this instance.
- by default, this method uses the `==` operator.
- has to be overriden to be useful.

```
public class MyDate
{   private int month, day, year; /*not a good coding practice */
                :
    @Override
    public boolean equals (Object obj)
    {   MyDate date = (MyDate) obj;
        if (date == null) return false;
        return date.getMonth() == getMonth ()
        && date.getDay() == getDay ()
        && date.getYear() == getYear ();
    }
}
```

equals()

- signature: `public boolean equals (Object obj)`
- tells whether some other object `obj` is *equal* to this instance.
- by default, this method uses the `==` operator.
- has to be overriden to be useful.

```
public class MyDate
{   private int month, day, year; /*not a good coding practice */
                :
    @Override
    public boolean equals (Object obj)
    {   // MyDate date = (MyDate) obj;
        if (date == null) return false;
        return date.month == month
                //date.getMonth() == getMonth ()
        && date.day == day  //date.getDay() == getDay ()
        && date.year == year;  //date.getYear() == getYear ();
    }
}
```

Type Casting

Type Casting is the explicit conversion from one data type to another.

Type Casting

Type Casting is the explicit conversion from one data type to another.

- The `obj` parameter is of type `Object`.

Type Casting

Type Casting is the explicit conversion from one data type to another.

- The `obj` parameter is of type `Object`.
- The attributes `month`, `day`, `year`, and the getter methods are not in `Object`.

Type Casting

Type Casting is the explicit conversion from one data type to another.

- The `obj` parameter is of type `Object`.
- The attributes `month`, `day`, `year`, and the getter methods are not in `Object`.
- All reference types are also `Object` type.

Type Casting

Type Casting is the explicit conversion from one data type to another.

- The `obj` parameter is of type `Object`.
- The attributes `month`, `day`, `year`, and the getter methods are not in `Object`.
- All reference types are also `Object` type.
- `(MyDate) obj` converts `obj` to a `MyDate` instance.

☺ **Thank you!** ☺