CS133: Developing Programming Principles

Lecture 9

All about `static`:
`static` methods, `static` variables,
`main` methods, constants,
wrapper classes

Warning! Warning! Danger!

- You will *use* **`static`** methods sometimes:
  - **`Math`** class
  - **`main`** methods
  - Wrapper classes
- You will almost never ***write*** **`static`** methods
- You will almost never *use* or *create* **`static`** variables (except for constants)

Don't overuse static!

`static` methods

- Belong to the class as a whole, not specific instances (objects) of it.
- Can be accessed without instantiating an object of the class type.

Temperature conversions

```
// C == Celsius, F == Fahrenheit, K == Kelvin
public class Temperature
{
  public static double fToC(double fahrenheitTemp)
  {
     return (fahrenheitTemp – 32.0)*5.0/9.0;
  }
  public static double cToK(double celsiusTemp)
  {
     return celsiusTemp + 273.15;
  }
  public static double fToK(double fahrenheitTemp)
  {
     double degCelsius = fToC(fahrenheitTemp);
     return cToK(degCelsius);
  }
  // Similarly, write cToF(), kToC(),and kToF()
}
```

Calling a **`static`** method

- Syntax:
  ***ClassName*`.`*Method*`(`*Arguments*`);`**
- Example:

```
System.out.println(Temperature.fToC(-40.0));
```
Result: _____

```
System.out.println(Temperature.cToK(21.0));
```
Result: _____

```
System.out.println(Temperature.fToK(85.0));
```
Result: _____

Invoking a non-**`static`** method in a **`static`** one

- **`static`** methods don't depend on instantiated objects.
- Instantiating an object makes space in memory for its instance variables.
- Thus **`static`** methods can't use instance variables.
- Thus, we can't call non-**`static`** methods from **`static`** ones directly (without instantiating an object of the class).
  - Why? Because non-**`static`** methods can use instance variables.

`static` variables

- Like methods, variables can be **`static`** .
- For example, within a class, some variables can be **`static`** .

`static` variables continued

Why?

Want some constant that is easily accessible

```
public static final double TAX_RATE = 0.15;
```

Sets this as a constant
(value cannot be changed)

Allow objects of the same class to share information. (**Rare**)

```
private static int numConversions = 0;
```

Board class constants

- The **Board** class also has **static** variables. Mostly constants.
- Examples:

```
public static final Color YELLOW = Color.YELLOW;
public static final Color BLACK = Color.BLACK;
public static final Color WHITE = Color.WHITE;
public static final Color RED = Color.RED;
public static final Color ORANGE = Color.ORANGE;
```

- Usage:

```
Board checkers = new Board(10, 10);
checkers.putPeg(Board.YELLOW, 3, 3);
```


static Variables: **BankAccount**

```
public class BankAccount {
  private static int numDeposits = 0;
  private static int nextAccountNumber = 1000;

  private double balance;
  private int accountNumber;

  public BankAccount(double initialBalance)
  {
     this.balance = initialBalance;
     this.accountNumber = nextAccountNumber;
     nextAccountNumber++;
  }
```

BankAccount continued

```
  public static void displayNumDeposits()
  {
     System.out.println("Deposits so far:" +
         numDeposits);
  }
  public void deposit(double amount)
  {  numDeposits++;
     this.balance += amount;
  }
  public int getAccountNumber()
  {  return this.accountNumber;
  }
  public double getBalance()
  {  return this.balance;
  }
  // More methods not shown (like withdraw())
} // end class
```


Using **BankAccount**

```
(a) BankAccount.displayNumDeposits();
(b) BankAccount first = new BankAccount(20.0);
(c) BankAccount second = new BankAccount(100.0);
(d) System.out.println(first.getAccountNumber());
(e) System.out.println(second.getAccountNumber());
(f) first.deposit(15.0);
(g) BankAccount.displayNumDeposits();
(h) second.deposit(20.0);
(i) first.deposit(10.0);
(j) first.displayNumDeposits();
(k) BankAccount third = new BankAccount(50.0);
(l) third.deposit(100.0);
(m) BankAccount.displayNumDeposits();
(n) System.out.println(third.getAccountNumber());
```

main methods

- Recall the first Java example:

```
public class MyFirstProgram
{
  public static void main(String[] args)
  {
    System.out.println("Hello world!");
  }
}
```

- Notice the **main** method is **static**.
  - Can call **main** by ***ClassName*.main(*Arguments*);**
  - If a class has a **main** method it must be **static**.
  - The **main** method cannot invoke non- **static** methods in that same class (without creating an instance of the object).

main methods continued

- Say we added a **main** method to the **BankAccount** class:

```
public class BankAccount {
  // same methods and variables as before
  public static void main(String[] args) {
     BankAccount.displayNumDeposits();
     BankAccount acct = new BankAccount(100.0);
     acct.deposit(200.0);
     System.out.println(this.balance); // BAD
     System.out.println(acct.balance); // OK!
  }
}
```

- This can be used to test methods

Wrapper classes

- We have seen:
  - Primitive types: int, char
  - Objects: **`Board`**, **`String`**, **`Calculator`**
- In fact, Java is **object-oriented**.
- So primitive types can actually be **wrapped** in an object.

Primitive	Wrapper class
`int`	`Integer`
`char`	`Character`
`double`	`Double`
`boolean`	`Boolean`

Wrapper classes: how they work

```
Integer num = new Integer(7);
System.out.println(num.intValue());
System.out.println(Integer.MAX_VALUE);
                //static variable
System.out.println(Character.isLetter('7'));
               //static method
int value = Integer.parseInt("56");
              //useful static method
```

Why have wrapper classes?

- Produces objects that correspond to primitive types
  - Now all data can be viewed as an object.
- Have **`static`** methods and **`static`** variables (constants) that may be useful.

Summary

- **`static`** methods
- Constants
- Using **`main`** methods
- Wrapper classes
- **`static`** variables

ERROR: undefined
OFFENDING COMMAND:

STACK: