Recap: Method Overloading
- Can have multiple methods of same name
- Distinguishes between them with signature
- Method name, parameter types and order
- Cannot have two methods with same signature
- Return type is not part of signature
- Any method can be overloaded
- Constructors are very common case

Recap: Interfaces
- Interface is collection of constants and abstract methods
- Different meaning than set of public methods that are documented, as in API
- To implement interface must provide definitions for all its methods
- Abstract methods have no implementation or body
- Method header followed by semicolon
- Specifies how to communicate with method, not what it does

Recap: Polymorphism
- Polymorphism: behavior varies depending on actual type of object
- Variables can be declared with interface as type, can invoke interface methods on them
- Cannot construct interface
- Can only construct objects of some particular class that implements interface
- Polymorphism determined at runtime
- Vs. method overloading, determined at compilation

Recap: Polymorphism Example

```
public class CokeMachine2005 implements VendingMachine {
    public void vendItem();
    public int getItemsRemaining();
    ...
    public double getCashReceived();
    public void loadItems(int n);
}
```

Recap: Interface Example

```
public class Comparable {
    public static void main(String[] args) {
        MyData myData;
        myData = new CokeMachine2005();
        myData.vendItem();
        myData.getItemsRemaining();
        myData.getCashReceived();
    }
}
```

Comparable
- Sort method that works on array of objects of any type that implements Comparable
- Type guaranteed to have compareTo method

Selection Sort For Int Primitives

```
1. selection sort
2. public static void selectionSort (int[] myArray) {
3.     int minIndex = myArray[0];
4.     for (int i = 0; i < myArray.length; i++) {
5.         if (myArray[i] < minIndex) {
6.             minIndex = myArray[i];
7.         }
8.     }
9.     for (int i = 0; i < myArray.length - 1; i++) {
10.        for (int j = 0; j < myArray.length - 1; j++) {
11.            if (myArray[j] > myArray[j+1]) {
12.                temp = myArray[j];
13.                myArray[j] = myArray[j+1];
14.                myArray[j+1] = temp;
15.            }
16.        }
17.    }
18.}
```

News: Reminder
- Change for labs
  - Week 11 was no lab. Now will be optional mid-term review/correction
  - A chance to work through your mistakes and get some marks back
  - People with Monday (holiday) labs or conflicts can attend another lab and/or work on their own. Anyone bring in corrected mid-term at beginning of the week 12 lab if not finished working through during week 11 lab

Reading
- Weeklies due today for this week
- Next week:
  - 8.1-9 (3rd ed)
  - 9.1-9.9 (2nd ed)

Interfaces as Contract
- Can write code that works on anything that fulfills contract
- Even classes that don’t exist yet!
- Example: Comparable
  - Useful if you need to sort items
  - compareTo(object)
    - Returns int < 0 if this object less than parameter
    - Returns 0 if same
    - Returns int > 0 if this object greater than parameter

Multiple Interfaces
- Classes can implement more than one interface at once
- Contract to implement all abstract methods defined in every interface it implements

Wrappers
- Many classes implement Comparable interface
  - Byte, Character, Double, Float, Integer, Long, Short, String
  - Each implements own version of compareTo
- Wrapper classes
  - Wraps up (encapsulates) primitive type
  - Double: object wrapping primitive double
    - NO: sort(double[] myData);
    - YES: sort(Double[] myData);