USING JAVA OBJECTS

- Objects and References
- The String Class
- Using Packages
- The Random Class
- The Math Class
- Comparing Floats, Strings, Objects
- Wrapper Classes
What happened to Comair Computers?

'Worst Christmas ever' as Comair grounds flights

December 26, 2004

HEBRON, Ky. – Comair canceled all its 1,100 flights Saturday because computer problems knocked out its system that manages flight assignments, a spokesman said.

Nick Miller, a spokesman for the Delta subsidiary based at Cincinnati/Northern Kentucky Regional Airport, said the cancellations affected 30,000 travelers in 118 cities.

"It's the worst Christmas I've ever experienced," said Guy Lobuono, who with his wife had been trying since Thursday to get from the Cincinnati airport to Wisconsin to visit their daughter's family. "We've missed Christmas Eve. We've missed Christmas."

Miller said the company was trying to put travelers on Delta flights. Crews were working to see how many flights Comair could handle today, but nothing was definite.

Miller said the problem was triggered partly by flights canceled Thursday and Friday because of bad weather.

"There was a cumulative effect with the canceled flights and trying to get crew assigned that caused the system to be overwhelmed," he said. "It just stopped operating."

Guy and Claire Lobuono, of Colerain Township near Cincinnati, spent five hours standing in a ticket line Thursday, only to find out their flight was canceled. Their flight was canceled again Friday, and they spent Christmas Eve at an airport hotel eating potato chips out of a vending machine because no restaurants were open.

"I know it's bad weather, but I just think it's disorganized," Lobuono said glumly, as his wife stood in line to check on other options.
There are eight primitive data types in Java

Four of them represent integers
  * byte, short, int, long

Two of them represent floating point numbers
  * float, double

One of them represents characters
  * char

And one of them represents logical values
  * boolean
2.3 – Numeric Primitive Data

The difference between the various numeric primitive types is their size, and therefore the values they can store:

<table>
<thead>
<tr>
<th>Type</th>
<th>Storage</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>8 bits</td>
<td>-128</td>
<td>127</td>
</tr>
<tr>
<td>short</td>
<td>16 bits</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>&lt; -9 x 10^{18}</td>
<td>&gt; 9 x 10^{18}</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>+/- 3.4 x 10^{38} with 7 significant digits</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>+/- 1.7 x 10^{308} with 15 significant digits</td>
<td></td>
</tr>
</tbody>
</table>

What happens when you mix different primitive types in a calculation?
2.5 – Assignment Conversion

What happens when you mix different primitive types in a calculation?

- Assignment conversion occurs when a value of one type is assigned to a variable of another

```c
float money;
int dollars;
```

- The following converts the value in `dollars` to a float

```c
money = dollars;
```

- Note that the value or type of `dollars` did not change

- Only widening conversions can happen via assignment

```c
dollars = money;
```

- In this case you cast: `dollars = (int)money;`
2.5 – Casting

- **Casting** is the most powerful (and dangerous), technique for conversion

- Both widening and narrowing conversions can be accomplished by casting a value

- To cast, put type in parentheses in front of the value being converted

- For example, if `total` and `count` are integers, but we want a floating point `result` when multiply them, we can cast

  ```
  result = (float) total * count;
  ```
2.5 – Promotion

*Promotion* happens automatically when operators in expressions convert their operands

```c
float total;
int count;
```

The value of `count` is converted to a floating point value to perform the following calculation

```c
result = total / count;
```
How do you program this?

- Your answer was: 123.456
- The correct answer was: 123.4555555555
- Your answer is correct!
4.3 – Comparing: Floats and Doubles

- When comparing two floating point values (float or double) you should **never** use the equality operator (==)

- Two floating point values are equal only if their underlying binary representations match exactly

- To determine the equality of two floats, you may want to use the following technique

  ```java
  if (Math.abs(f1 - f2) < TOLERANCE)
      System.out.println("Essentially equal");
  ```

- If the difference between the two floating point values is less than the **TOLERANCE**, they are considered to be equal

- **TOLERANCE** could be set to an appropriate level, i.e. 0.001
4.3 – Comparing: chars

- Java characters are based on the **Unicode** character set.

- Unicode establishes a particular numeric value for each character, and therefore an **ordering**.

- For example, the character '+' is less than 'J' because it comes before it in the Unicode character set.

- the digit characters (0-9) are **contiguous** and in **order**.

- the uppercase letters (A-Z) and the lowercase letters (a-z) are **contiguous** and in **order**.

<table>
<thead>
<tr>
<th>Characters</th>
<th>Unicode Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 – 9</td>
<td>48 through 57</td>
</tr>
<tr>
<td>A – Z</td>
<td>65 through 90</td>
</tr>
<tr>
<td>a – z</td>
<td>97 through 122</td>
</tr>
</tbody>
</table>
4.3 – Comparing: Strings

The `equals` method can be called with strings to determine if two strings contain exactly the same chars in the same order.

The `compareTo` can determine their lexicographical order.

```java
if (name1.equals(name2))
    System.out.println("Same name");
else if (name1.compareTo(name2) < 0)
    System.out.println(name1 + "comes first");
else if (name1.compareTo(name2) == 0)
    System.out.println("Same name");
else
    System.out.println(name2 + "comes first");
```

Which comes first, "Great" or "fantastic"? "book" or "bookcase"?
4.3 – Comparing: Objects

- The `==` operator can be applied to objects – it returns true if the two references are aliases of each other.

- The `equals` method is defined for all objects, but unless we redefine it when we write a class, it has the same semantics as the `==` operator.

- Avoid using `==` even for the `String` class; better use the `equals` method.

- When you write a class, you should redefine the `equals` method to return true under whatever conditions are appropriate.
3.8 – Wrapper Classes

- Are the primitive types objects?
- The java.lang package contains *wrapper classes* that correspond to each primitive type

<table>
<thead>
<tr>
<th>Primitive Type</th>
<th>Wrapper Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte</td>
<td>Byte</td>
</tr>
<tr>
<td>short</td>
<td>Short</td>
</tr>
<tr>
<td>int</td>
<td>Integer</td>
</tr>
<tr>
<td>long</td>
<td>Long</td>
</tr>
<tr>
<td>float</td>
<td>Float</td>
</tr>
<tr>
<td>double</td>
<td>Double</td>
</tr>
<tr>
<td>char</td>
<td>Character</td>
</tr>
<tr>
<td>boolean</td>
<td>Boolean</td>
</tr>
<tr>
<td>void</td>
<td>Void</td>
</tr>
</tbody>
</table>
3.8 – Wrapper Classes

- An `Integer` object which represents the integer 40 as an object:
  ```java
  Integer age = new Integer(40);
  ```

- Wrapper classes contain static methods that help manage the type

- For example, the `Integer` class contains a method to convert an integer stored in a `String` to an `int` value:
  ```java
  num = Integer.parseInt(str);
  ```

- How about storing an Integer as a String?

- The wrapper classes often contain useful constants
  - For example, the `Integer` class contains `MIN_VALUE` and `MAX_VALUE` which hold the smallest and largest `int` values
  ```java
  System.out.println(Integer.MIN_VALUE);
  ```
  prints:
3.8 – Autoboxing and Unboxing

Autoboxing is the automatic conversion of a primitive value to a corresponding wrapper object.

```
Integer i;
int num = 42;
i = num;  // autoboxing of num
```

The assignment creates the appropriate Integer object.

The reverse conversion (unboxing) also occurs automatically as needed:

```
Integer j = Integer(50);
num = j;  // unboxing of j
```
2.6 – Getting Input: The **Scanner** Class

- The **Scanner** class provides convenient methods for reading input values of various types.
- A **Scanner** object can read input from various sources.
  - E.g., **Keyboard** input is represented by the **System.in** object:

```java
Scanner scan = new Scanner(System.in);
```
- Once created, the **Scanner** object can be used to invoke various input methods, such as

```java
String message = scan.nextLine();
```
- The `nextLine` method reads all of the input until the “end of the line” character is found.
Demonstrates the use of the `nextLine` method of the `Scanner` class to read a string from the user.
2.6 – Input Tokens

- Unless specified otherwise, *white space* is used to separate the input elements (called *tokens*).

- White space includes space characters, tabs, new line characters.

- The *next* method of the *Scanner* class reads the next input token and as a *String*.

- Methods such as *nextInt* and *nextDouble* read data of particular types.
import java.util.Scanner;

public class GasMileage {
    public static void main(String[] args) {
        int miles;
        double gallons, mpg;

        Scanner scan = new Scanner(System.in);

        System.out.print("Enter the number of miles: ");
        miles = scan.nextInt();

        System.out.print("Enter the gallons of fuel used: ");
        gallons = scan.nextDouble();

        mpg = miles / gallons;
        System.out.println("Miles Per Gallon: "+ mpg);
    }
}