CSCI 161
Introduction to Computer Science
Outline

- Data Types
  - Primitives: ints, doubles, chars, booleans, etc.
    - Type Casting
    - Boolean Expressions
  - Classes and Object References
    - And What Is null?

- Conditionals Revisited
  - Object Equality: referential vs. content

- Conclusion
Recall: Every piece of data in Java (local variables, fields, input parameters) has a *data type*

Java exposes two kinds of *data types*:

1) Primitives: int, double, boolean, char, long, byte, short, float
   - E.g., `int x = 9;`

2) Classes are also data types! String, Random, Circle, ...
   - Reference to an object stored inside a variable
   - E.g., `Triangle t = new Triangle();`
Integer Overflow

- Say we have an int variable...
  - What's the largest or smallest number it can store?
  - Is there a limit?

```java
// Try this
int x = 2147483647;
x += 1000;
System.out.println(x);
```
Range of Some Primitive Types

- **Range of primitive values:**
  
  - int range: \([-2^{31}, +2^{31} - 1]\), which is \([-2147483648, +2147483647]\)
  
  - long range: \([-2^{63}, +2^{63} - 1]\), which is little more than 9 with 18 zeros following
  
  - double range: \([-4.941 \times 10^{-324}, 1.7977 \times 10^{308}]\)
  
  - char values: 'a', 'A', 'b', 'B', '#', '@', ..
  
  - There are also: byte, short, and float
Operations (for ints and doubles)

- Integer (int, long, byte, short) Operators

<table>
<thead>
<tr>
<th>Add +</th>
<th>Subtract -</th>
<th>Multiply *</th>
<th>Divide /</th>
<th>Modulo %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>3 + 4 + 5</td>
<td>(0 - 3) - 1</td>
<td>4 * 5</td>
<td>40 / 6</td>
</tr>
<tr>
<td>Result</td>
<td>12</td>
<td>-4</td>
<td>20</td>
<td>6</td>
</tr>
</tbody>
</table>

- Real number (double, float) Operators

<table>
<thead>
<tr>
<th>Add +</th>
<th>Subtract -</th>
<th>Multiply *</th>
<th>Divide /</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>15.8 + 3.2</td>
<td>9.2 - 2.0</td>
<td>3.14 * 2</td>
</tr>
<tr>
<td>Result</td>
<td>19.0</td>
<td>7.2</td>
<td>6.28</td>
</tr>
</tbody>
</table>

- Also, know these accumulators:

```
x++;    x--;    x += y;    x -= y;    x *= y;    x /= y;
```
After you declare a variable, you can assign a value to it.

- But what is inside the variables before you assign their values?
  - Number types default to 0, boolean defaults to false

```java
int x;
double y;
boolean z;
x > 0
y
> 0
z
> false
```

What's being stored in your computer's memory?

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
<th>Data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>(int)</td>
</tr>
<tr>
<td>y</td>
<td>0.0</td>
<td>(double)</td>
</tr>
<tr>
<td>z</td>
<td>false</td>
<td>(boolean)</td>
</tr>
</tbody>
</table>
Won't compile when you assign *wrong* values to variables

- Here, the value of `x` (an int) is not in `z`'s *range of true or false*

```java
int x;
double y;
boolean z;
x = 90;
y = x;  // this is okay, despite type-mismatch
z = x;  // this causes the following compiler error

> Error: incompatible types: int cannot be converted to boolean
```

*Is the following type-mismatch okay?*

```java
int x;
double y = 90;
x = y;

> Error: incompatible types: possible lossy conversion from double to int
```
Outline

- Data Types
  - Primitives: ints, doubles, chars, booleans, etc.
    - Type Casting
    - Boolean Expressions
  - Classes and Object References
    - And What Is null?

- Conditionals Revisited
  - Object Equality: referential vs. content

- Conclusion
Type Casting (Conversion)

- We can also ask Java to convert numerical types between each other.
  
  • **Widening-conversion (Done automatically)**
    
    - Convert a "smaller" type to a "bigger" type
    
    - byte --> short --> int --> long --> float --> double
  
  • **Narrowing-conversion (Done explicitly by programmer)**
    
    - Convert a "bigger" type to a "smaller" type
    
    - double --> float --> long --> int --> short --> byte
Narrowing Conversion

- Narrowing Conversion Syntax:

  Assume x is declared as `narrowerType`

  `x = (narrowerType) expr;`

  `expr` is any expression resulting in a wider type.

- Examples:

  ```java
  int x;
  double y = 90.25;
  x = (int) (y * 100);  // x holds 9025
  ```

  ```java
  int x;
  double y = 90.25;
  x = (int) y * 100; // x holds 9000 (why?)
  ```
Mixing Types in Math Expressions

- **Important**: If there are mixed types in an expression, the result's data type is the type with the highest precision!

```java
int x = 2;
double y = 7.50;

int z = x * y;

> Error: incompatible types: possible lossy conversion from double to int
```
### Examples of Numerical Operations

```c
int iResult, num1 = 25, num2 = 40, num3 = 17, num4 = 5;
double fResult, val1 = 17.0, val2 = 12.78;

iResult = num1 / num4;  // #0
> 5
fResult = num1 / num4;  // #1
> 5.0
iResult = num3 / num4;  // #2
> 3
fResult = num3 / num4;  // #3
> 3.0
iResult = num3 % num4;  // #4
> 2
fResult = val1 / num4;  // #5
> 3.4
fResult = (double) num1 / num2;  // #6
> 0.625
fResult = (double) (num1 / num2); // #7
> 0.0
fResult = (int) ((double) num1 / num2); // #8
> 0.0
```
Outline

- **Data Types**
  - Primitives: ints, doubles, chars, booleans, etc.
    - Type Casting
    - Boolean Expressions
  - Object Types and References
    - And What Is null?

- **Conditionals Revisited**
  - Object Equality: referential vs. content

- **Conclusion**
Combining Conditionals

- Conditionals can also be combined using: &&, ||, or !
  - condition1 && condition2 (called an **AND**)
  - condition1 || condition2 (called an **OR**)
  - !condition (called a **NEGATION** or a **NOT**)

- Check if x is either 1, 2, or 3

  *(Which of these statements are correct?)*

```
if (x > 0 || x <= 3) {
  ...
}
```

WRONG

```
if (x > 0 && x <= 3) {
  ...
}
```

CORRECT

```
if (x == 1 && x == 2 && x == 3) {
  ...
}
```

WRONG

```
if (x == 1 || x == 2 || x == 3) {
  ...
}
```

CORRECT
Your Turn! Largest of Three

- Write a method:
  - `maxOfThree` that inputs 3 integers and returns the largest.

```java
public int maxOfThree(int x, int y, int z) {
    // Your turn!
}
```
Outline

- Data Types
  - Primitives: ints, doubles, chars, booleans, etc.
    - Type Casting
    - Boolean Expressions
  - Classes and Object References
    - And What Is null?

- Conditionals Revisited
  - Object Equality: referential vs. content

- Conclusion
Primitive Type Variables vs. Object Variables

- So we know primitive variables hold specific values

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>(int)</td>
</tr>
<tr>
<td>y</td>
<td>0.0</td>
<td>(double)</td>
</tr>
<tr>
<td>z</td>
<td>false</td>
<td>(boolean)</td>
</tr>
</tbody>
</table>

- What about object variables?

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value</th>
<th>Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>??</td>
<td>(Triangle)</td>
</tr>
<tr>
<td>calc</td>
<td>??</td>
<td>(Calculator)</td>
</tr>
</tbody>
</table>
So we know primitive variables hold specific values

Object variables? They store *references (or pointers) to all the data representing an object.*

```
Calculator calc = new Calculator();
// How is the 'calc' variable
// represented inside
// a computer?
```
Objects: null Values

- What are the default values for objects?
  - Notice, I'm not instantiating any objects below

```java
Triangle myTriangle;
Square body;
String name;

System.out.println(myTriangle + "", " + body + ", " + name);
//what gets printed here?
```

- Output in the BlueJ Terminal Window:
  ```
  null, null, null
  ```

- **Important:** `null` is Java's "I know to nothing" keyword for objects
Big Differences in Assignment Operator

Know this! When *assigning* values to another variable:

- Primitive variables get a **cloned** copy!

```java
int x = 10;
int y = x;
```

- Object assignment **doesn't clone** objects! Only the arrow is updated.

```java
String x = "David";
String y = x;
```

```
x (int)  y (int)
10      10
```

```
x (String)  y (String)
"David"    "David"
```
What Do Object Variables Hold?

- Object variables hold a *reference* that points to the data representing the object.

```java
Calculator calc; //calc can hold a reference to a Calculator
```

```
calculator
null
```

```java
calculator (Calculator)
```
Object variables hold a \textit{reference} that points to the data representing the object.

```java
Calculator calc; //calc can hold a reference to a Calculator
calc = new Calculator(); // instantiate a calculator that `calc` points to
```
Object variables hold a *reference* that points to the data representing the object.

```java
Calculator calc;  // calc can hold a reference to a Calculator
calc = new Calculator();  // instantiate a calculator that
            // `calc` points to
calc = null;
```
Object variables hold a *reference* that points to the data representing the object.

```java
Calculator c1;
c1 = new Calculator();

Calculator c2;
c2 = new Calculator();
```
Object variables hold a *reference* that points to the data representing the object.

```java
Calculator c1;
c1 = new Calculator();

Calculator c2;
c2 = new Calculator();

c1 = c2;  // The Calculator object originally referenced by c1 is lost forever
```
Object variables hold a *reference* that points to the data representing the object.

```java
Calculator c1;
c1 = new Calculator();

Calculator c2;
c2 = new Calculator();

c1 = c2;

c2 = null; // The Calculator is not lost because it is still being referenced
```
What Do Object Variables Hold?

- Object variables hold a *reference* that points to the data representing the object.

```java
Calculator c1 = new Calculator();
Calculator c2 = c1;  // Refer to the same card.
                   // Objects are not cloned!

c1.add(50);  // Same as c2.add(50)
c2.multiply(10); // Same as c1.multiply(10)

System.out.println(c1.getValue());  // prints 500
System.out.println(c2.getValue());  // prints 500
```
Outline

- Data Types
  - Primitives: ints, doubles, chars, booleans, etc.
    - Type Casting
    - Boolean Expressions
  - Classes and Object References
    - And What Is null?

- Conditionals Revisited
  - Object Equality: referential vs. content

- Conclusion
Comparing Primitives vs. Comparing Objects

- **Comparing 2 primitive values** inside if-statements:
  - We use comparison operators (==, !=, <=, >=, <, >)
  - Recall that comparison operators result in a boolean value (true, false)

- What about **comparing 2 objects**?
  - What do the comparison operators even mean in these cases?
    - What does it mean to ask if one Circle < String?
      - You would never want to do this!
    - What does it mean to ask if one Calculator >= another Calculator?
      - Up for interpretation by the programmer... (foreshadowing)
Most comparison operators (\(<=\), \(\geq\), \(<\), \(>\)) don't make sense when used to compare objects in an if-statement.

- In fact, only `==` and `!=` can be used to compare object variables

For example

```java
Circle A = new Circle();
Circle B = A;

// .. more code omitted..

if (A == B) {
    // Do A and B point to the same Circle?
}
```
Reference Equality of Objects

- **Reference Equality**: Are 2 variables *pointing to* the same object?
  - Can only be checked using `==` and `!=`

```java
Triangle T = new Triangle();
Circle C = new Circle();
if (T != C) {
    // This statement triggers!
}
```
Testing for NULL

- Most commonly, the `==` and `!=` operators are used to check if an object variable points to nothing (`null`)

```java
Circle A;
if (A == null) {
    // this code will get executed!
}

Circle B = A;
if (A == B) {
    // so will this code
}
```

```mermaid
graph TD
A[Circle] --> null
null --> B[Circle]
```
Testing for NULL

- The NULL check is often used for *defensive programming*

```java
public void doStuff(Circle c) {
    // what if user inputs a null for c?
    c.changeSize(100);
    c.changeColor("pink");
}
```

```
null
(Circle)
```
Testing for NULL

- The NULL check is often used for *defensive programming*

```java
public void doStuff(Circle c) {
    // what if user inputs a null for c?
    c.changeSize(100);
    c.changeColor("pink");
}
```

Hey pal, can you change your size to 100?

I know nothing! Crash the program.

Called a `NullPointerException`
The NULL check is often used for *defensive programming*

```java
public void doStuff(Circle c) {
    // No more NullPointerException errors causing program to crash!
    if (c != null) {
        c.changeSize(100);
        c.changeColor("pink");
    }
}
```
Outline

- Data Types
  - Primitives: ints, doubles, chars, booleans, etc.
    - Type Casting
    - Boolean Expressions
  - Classes and Object References
    - And What Is null?

- Conditionals Revisited
  - Object Equality: reference vs. content

- Conclusion
Is Reference Equality Sufficient?

Consider this next example.

```java
Square sq1 = new Square();
sq1.changeSize(10);
sq1.changeColor("black");

Square sq2 = new Square();
sq2.changeSize(10);
sq2.changeColor("black");

//this statement is false, because they refer to different objects
if (sq1 == sq2) {
    //does this get executed?
}
```

```
\[sq1 \hspace{1cm} \text{(Square)}\]
\[\text{10}\]
\[\text{10}\]

\[sq2 \hspace{1cm} \text{(Square)}\]
\[\text{10}\]
\[\text{10}\]
```
Consider another example (with Strings):

```java
String str1 = "Puget Sound";
String str2 = "Puget Sound";

if (str1 == str2) {
    // However, once again, this statement will be false!!!
}
```
New problem: Reference equality only tests the variables' references!

- It does not check the content of objects

An Object's Content Equality:

- If we need to determine if two objects' contents are equal, we need to:

  Step 1: YOU decide what it means for two objects of the same class to be equal in content. (Usually, some set of their states must be equal).

  Step 2: Provide a public boolean equals(...) method for the class that performs the check. (Provide getters, if necessary)

  Step 3: Test using, obj1.equals(obj2)
Content Equality of Squares

- We'll use Squares as the running example:
  - Open up ``Shapes'' project

Steps:

- Step 1: Look at the state for Squares; decide what it would mean for two Squares to be equal.
- Step 2:
  - Do we have a way to get all that information? (Provide getters when applicable)
  - Write `public boolean equals(Square other)`
- Step 3: Test it!
Now this works!

```
Square sq1 = new Square();
Square sq2 = new Square();
sq1.changeSize(10);
sq2.changeSize(10);
sq1.changeColor("black");
sq2.changeColor("black");

//now this statement is true, because they are equal in size
if (sq1.equals(sq2) == true) {
    // This will now trigger!
}
```
Object-Equivalence Example: OrcaCard

- Coders decide: "What needs to be true for two OrcaCards to be equal?"

```java
public class OrcaCard {
    private double balance; // The current balance
    private double tax; // Amount of taxes collected

    /**
     * Checks another OrcaCard for equality
     * @param other_card
     * @return true if equals, false otherwise
     */
    public boolean equals(OrcaCard other_card) {
        return (this.balance == other_card.getBalance() &&
                this.tax == other_card.getTax());
    }
}
```
Content Equality of Strings

- String content-equivalence methods are provided in the String class!
  - How they defined equivalence (two options):

  Provided by the String class
  ```java
  public boolean equals(String s)
  public boolean equalsIgnoreCase(String s)
  ```

- Example:

  ```java
  String name1 = "University of Puget Sound";
  String name2 = "uNiVeRsItY oF pUgEt SoUnD";
  if (name1.equals(name2)) {
    // doesn't fire
  }
  ```
String content-equivalence methods are provided in the String class!

• How they defined equivalence (two options):

  Provided by the String class
  
  ```java
  public boolean equals(String s)
  public boolean equalsIgnoreCase(String s)
  ```

Example:

```java
String name1 = "University of Puget Sound";
String name2 = "uNiVeRsItY oF pUgEt SoUnD";

if (!name1.equals(name2)) {  // Asks: are the strings not equal (case sensitive)?
   // This fires
}
```
String content-equivalence methods are provided in the String class!

- How they defined equivalence (two options):

```java
String name1 = "University of Puget Sound";
String name2 = "uNiVeRsItY oF pUgEt SoUnD";

if (name1.equalsIgnoreCase(name2)) {
    // This fires!
}
```
Main Takeaways

- The operators `==` and `!=` are only used to check reference equality
  - Usually used to check if an object is null
  - Know what a NullPointerException is
    - (And how to prevent them)

- When writing classes
  - **Always** define an `equals(...)` method for checking content equality
  - Totally up to you to determine what "equals" means in your class
Hwk 2 due Friday
  • Questions?

Lab tomorrow (Guessing Game):
  • Emphasis: the Random object, but mostly conditionals

Last time...
  • We practiced more on if-then-else, else-if

Today...
  • Data types: primitive types vs. classes
  • Mixed-mode expressions
  • Type casting (conversion)
Hwk 2 (Calculator) due tonight

Lab 4 (Guessing Game) post-mortem

- Good:
  - Generating random numbers, writing abs() method

- Struggles:
  - Flow of execution (knowing *where* to put things)
    - Tip: Step back, use paper pencil to plan out algorithm in English before writing code
  - Coding style is more about correctness, not prettiness!
    - Tip: When you open a brace, close it right away.
    - Tip: Indenting your code properly is so important!!!!!!
      » Use auto-indent feature!!
Last time...

- Learned all about **primitive** data types:
  - Type casting (conversion) between types: widening vs narrowing
  - Overflow of numerical types (Ariane rocket explosion, Therac-25 radiation machine)
  - Mixed-type expressions

Today...

- One more primitive: boolean, and its operators
- **Classes** are also data types
- But their usage and representation are totally different from primitives
Hwk 3 (OrcaCard) due next Wednesday

Exam next Thursday
  • Start casually looking through Review Guide 1

Last week...
  • Talked about strategies for keeping code organized in BlueJ
  • All about boolean expressions: &&, ||, !
    - Finishes up primitive data types

Today:
  • Classes as data types too!
  • Object references
  • Object equality: reference equality vs. content equality