CSCI 161
Introduction to Computer Science

Lecture 3
Data Types: Primitives vs. Object Types
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
Recall: Every piece of data in Java (local variables, fields, input parameters) are declared with a particular data type.

Java knows two different kinds of data types:

- Primitive types: int, double, boolean, char, long, ...
  - Value stored directly inside a variable
  - E.g., `int x = 9;`

- Classes: String, Random, Circle, ...
  - Reference to an object stored directly 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);
```

---

Ariane 5 Rocket Explosion

Therac 25
Range of Some Primitive Types

- First, we'll focus on primitives

- Range of primitives:
  - `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: \([-4.941 \times 10^{-324}, 1.7977 \times 10^{308}]\)
  - `boolean` range: `true`, `false`
  - `char` range: `'a'`, `'A'`, `'b'`, `'B'`, '#', '@', ...
Operations (for ints and doubles)

- Integer \((\text{int})\) Operators

<table>
<thead>
<tr>
<th></th>
<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>
<td>40 % 6</td>
</tr>
<tr>
<td>Result</td>
<td>12</td>
<td>-4</td>
<td>20</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

- Real number \((\text{double})\) Operators

<table>
<thead>
<tr>
<th></th>
<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>
<td>5.0 / 3.0</td>
</tr>
<tr>
<td>Result</td>
<td>19.0</td>
<td>7.2</td>
<td>6.28</td>
<td>1.6666666...7</td>
</tr>
</tbody>
</table>

- Also, know these \textit{accumulators}:

\begin{verbatim}
int x; 

x++;  // x = x + 1
x--;  // x = x - 1
x += y;  // x = x + y
x -= y;  // x = x - y
x *= y;  // x = x * y
x /= y;  // x = x / y
x %= y;  // x = x % y
\end{verbatim}
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>Data type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>(int)</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>(double)</td>
<td>0.0</td>
</tr>
<tr>
<td>z</td>
<td>(boolean)</td>
<td>false</td>
</tr>
</tbody>
</table>
Primitive Values (Cont.)

- What happens when you assign *wrong* values to certain variables?
  - Here, the value of \( x \) is not in \( z \)'s range of true or false

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

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

- Another example:

```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

- Object Types and References
  - And What Is null?

Conditionals Revisited

- Object Equality: referential vs. content

Conclusion
Type Casting (Conversion)

- But what if I actually wanted to remove the fraction?
  - Need to tell Java that!
  - Syntax:

> x is declared as `targetType`

expr is any expression resulting in higher precision than target type

> x = (targetType) expr;

**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

```java
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
```
What about Booleans?

- *Comparison operators* (below) return a **boolean** value
  
  - Below, `a` and `b` are variables or expressions that result in an **int** or **double**

<table>
<thead>
<tr>
<th>Comparison Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a == b</code></td>
<td>Equals?</td>
<td><code>x == (93-60)</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>a != b</code></td>
<td>Not equals?</td>
<td><code>4/5 != 1</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>a &lt;= b</code></td>
<td>Less than equals?</td>
<td><code>3 &lt;= 2</code></td>
<td><code>false</code></td>
</tr>
<tr>
<td><code>a &gt;= b</code></td>
<td>Greater than equals?</td>
<td><code>3 &gt;= 2</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>a &lt; b</code></td>
<td>Less than?</td>
<td><code>-(13 * 8) &lt; -10</code></td>
<td><code>true</code></td>
</tr>
<tr>
<td><code>a &gt; b</code></td>
<td>Greater than?</td>
<td><code>2 &gt; 13</code></td>
<td><code>false</code></td>
</tr>
</tbody>
</table>
Example: Old Exam Question

Consider the code to the left. What values of \( z \) would produce the following results:

- \( X \)
- \( XX \)
- \( XXX \)
- \( XXXX \)
- No output

```java
if (z < 10) {
    System.out.println("X");
}
else if (z < 9) {
    System.out.println("XX");
}
else if (z > 15) {
    System.out.println("XXX");
}
else if (z == 10) {
    System.out.println("XXXX");
}
```
Consider the code to the left. What values of $z$ would produce the following results:

- $X$ --- $z < 10$
- $XX$ --- can't be produced
- $XXX$ --- $z > 15$
- $XXXX$ --- $z$ is 10
- No output --- $z$ is in $[11, 15]$
Write a method:

- `getNumOdd` that inputs two integers and returns the number of inputs that are odd: 0, 1, or 2.
  - (But how do you determine if an integer is odd or even?)

```java
public int getNumOdd(int n1, int n2) {
    // Implementation goes here
}
```
Combining Boolean Expressions

- Boolean expressions can be combined using:
  - `bool1 && bool2` (called an **AND**)
  - `bool1 || bool2` (called an **OR**)
  - `!bool` (called a **NEGATION** or a **NOT**)

- Check if `x` is in range `(0, 3]`  
  *(Which of these statements are correct?)*

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

WRONG

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

CORRECT

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

CORRECT

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

Depends...
Examples of Boolean Operations

```java
int value1 = 5, value2 = 10;
boolean done = true;

value1 < ((value2+1) / 2)   // 0
   > false

!(value1 + 5 != value2)    // 1
   > true

(value1 > value2) || done   // 2
   > true

(value1 < value2) && !done   // 3
   > false

done || !done               // 4
   > true

((value1 > value2) || done) && (!done || (value2 > value1))   // 5
   > true
```
Example: Another Old Exam Question

- Write a method:
  - `getNumOdd` that inputs two integers and returns the number of inputs that are odd: 0, 1, or 2.
  - (But how do you determine if an integer is odd or even?)

```java
public ______ getNumOdd(int n1, int n2) {
    // they are both odd
    if (n1 % 2 == 1 && n2 % 2 == 1) {
        return 2;
    }

    // neither of them is odd
    if (n1 % 2 == 0 && n2 % 2 == 0) {
        return 0;
    }

    // only one of them is odd
    return 1;
}
```
Your Turn! Largest of Three

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

```java
public _____ maxOfThree(_____________________) {
  // Your turn!
}
```
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
### 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>
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>

- Object variables? They store *references (or pointers) to all the data representing an object*.
Objects: null Values

- What are the default values for objects?

```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 point to nothing" keyword for objects
Object variables hold a *reference* that points to the data representing the object.

```java
OrcaCard card;  //card can hold a reference to an OrcaCard
```

![Diagram showing a variable named 'card' that points to a value of type 'OrcaCard', which is null.](image)
What Do Object Variables Hold?

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

```java
OrcaCard card;
card = new OrcaCard(0.09); // instantiate an OrcaCard that card points to
```

Field:
- balance: 0
- taxRate: 0.09
- ...

Methods:
- getTax()
- buyTrip()
- ...

What Do Object Variables Hold?

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

```java
OrcaCard card;
card = new OrcaCard(0.09);
card = null; // The OrcaCard is lost forever
```
Object variables hold a **reference** that points to the data representing the object.

```java
OrcaCard card1;
card1 = new OrcaCard(0.09);

OrcaCard card2;
card2 = new OrcaCard();
```
Object variables hold a *reference* that points to the data representing the object.

```java
OrcaCard card1;
card1 = new OrcaCard(0.09);

OrcaCard card2;
card2 = new OrcaCard();

card1 = card2; // The OrcaCard object originally referenced by card1 is lost forever
```
What Do Object Variables Hold?

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

```
OrcaCard card1;
    card1 = new OrcaCard(0.09);

OrcaCard card2;
    card2 = new OrcaCard();

    card1 = card2;

    card2 = null;  // The OrcaCard 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
OrcaCard card1 = new OrcaCard(0.09);
OrcaCard card2 = card1;  // Refer to the same card.
                         // Objects are not cloned!

card1.topUp(50.0);  // Same as card2.topUp(50.0)
card2.buyTrip(10);   // Same as card2.buyTrip(10)
```
Takeaway for Object Types vs. Primitive Types

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

  - Primitive variables get an exact cloned copy!

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

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

    ```
    String x = "David";
    String y = x;
    ```
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
Comparing Objects?

- Comparison operators (== != <= >= < >) don't always make sense when used with objects.
  - In fact, only == and != can be used to compare object variables
  - *(But they don't do what you might think they do....)*

- 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?
}
```
Referential Equality of Objects

- **Referential Equality**: Are 2 variables *pointing to* the same object?
  - Important: Checked using `==` and `!=`

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

![Diagram showing comparison between Triangle T and Circle C](image-url)
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
}
```

![Diagram showing the relationship between `A`, `B`, and `null`]
The NULL check is often used for *defensive programming*.

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

![Diagram showing a null pointer and a Square object](null-pointer-square)
The NULL check is often used for *defensive programming*.

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

Hey pal, can you change your size to 100?

I know nothing! Crash the program.

Called a NullPointerException.
Testing for NULL (Fix)

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

```java
public void doStuff(Square s) {
    // No more NullPointerException errors causing program to crash!
    if (s != null) {
        s.changeSize(100);
        s.changeColor("pink");
    }
}
```
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: reference vs. content

Conclusion
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?
}
```

![Diagram showing two squares, sq1 and sq2, with the same size and color, but different references](image)
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!!!
}
```

```
str1 (String) -> "Puget Sound"
str2 (String) -> "Puget Sound"
```
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)) {
    // 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 (balance == other_card.getBalance() && tax == other_card.getTax());
    }
}
```
String content-equivalence methods are provided in the String class!

- How they defined equivalence (two options):

  ```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):

<table>
<thead>
<tr>
<th>Provided by the String class</th>
</tr>
</thead>
<tbody>
<tr>
<td>public boolean equals(String s)</td>
</tr>
<tr>
<td>public boolean equalsIgnoreCase(String s)</td>
</tr>
</tbody>
</table>

**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 3 due next Monday (2/25)

Start preparing for midterm exam
  • Look over old code and homework/lab solutions
  • Practice writing code on pencil and paper
  • Look over review guide

Last time:

Today:
  • More on primitives
  • Focus on booleans
Hwk 3 due next Monday (2/25)

Reminder to start preparing for midterm exam
• Look over old code and homework/lab solutions
• Practice writing code on pencil and paper
• Look over review guide

Last time:
• Mixing types in expressions
• Type compatibility, type conversion / casting
• Wrote method: getNumOdd

Today:
• Focus on booleans
Hwk 3 due Monday (2/25)

Lab 5 post-mortem: (Solution posted)
  • Takeaway 1: Classes are data types too!
    - What are nulls? (Today)
  • Takeaway 2: Returning
    - What's the point of returning values in a method?
  • Takeaway 3: Code instead of menus
    - The static void main() method?
    - How do you "boss" an object around?

Today:
  • Object types, nulls, object equality