Today's topics

Java
  Input
  More Syntax

Upcoming
  Decision Trees
  More formal treatment of grammers

Reading
  Great Ideas, Chapter 2

Java Details

- **Variable**: an item of data named by an identifier
- **Constants (also called self-defining term)**
  - 42
  - 35.45
  - “Hello”
- **Operators**
  - Arithmetic
  - Relational and conditional
  - Assignment
  - Other
- **Expression**: a series of variables, operators, and method calls that evaluates to a single value

Syntax, Semantics, and Style

- **Syntax**
  - Grammar
  - Java requires it be perfect
  - Compiler will generate error messages
- **Semantics**
  - Meaning
  - Compiler will not (can not!) check most of it
  - You can write incorrect (or stupid, or inefficient) programs
- **Style**
  - Make program more readable for humans
  - Actually very important!
  - Helps understanding and writing correct programs

Data Input

- For program to be versatile it must have Input
- Have used Buttons as a form a input
  - It’s one way to make our wishes known
- Need more flexibility
  - Can input text: String
  - Can input numbers
    - Whole numbers called integers: int
    - Real numbers (allow fractions) called doubles: double
  - Use `getText` of TextField class to input strings
  - Use `getInt` of IntField class to input integers
Text (string) Input

- Use TextFields to read in string data
- Use the getText method of the TextField class
  - After creating a TextField object we can use method
  - Syntax (we've used it before) is
    - object.method()
  - For example note following code fragment:
    - // declare and create TextField instr
    - TextField instr = new TextField(50);
    - // declare message (new not needed)
    - String message;
    - // message gets value from TextField instr
    - message = instr.getText();

Text Input Example

```java
public class DupThree extends java.applet.Applet
    implements ActionListener
{
    TextField m1, m2, m3, m4, m5;
    Button b1;
    String message;
    public void init() {
        m1 = new TextField(80);
        m2 = new TextField(80);
        m3 = new TextField(80);
        m4 = new TextField(80);
        m5 = new TextField(80);
        b1 = new Button("button");
        add(m1); add(m2); add(b1); add(m3); add(m4); add(m5);
    }

    public void actionPerformed(ActionEvent event) {
        // since there is only one button, no if needed
        message = m2.getText();
        m3.setText(message);
        m4.setText(message);
        m5.setText(message);
    }
}
```

Text Input Example (continued)

```java
public void actionPerformed(ActionEvent event) {
    // since there is only one button, no if needed
    message = m2.getText();
    m3.setText(message);
    m4.setText(message);
    m5.setText(message);
}
```

Dealing with numbers

- **Primitive data type: int**
  - Does not require a new operator to create
  - Primitive type is not a class
  - Must declare
  - Should initialize
  - Other primitive types include: boolean, char, double

- **Operations using integers**
  - +, -, *, /, %
  - Operator Precedence
Some arithmetic details

- Java adheres to traditional order of operations
  - * and / have higher precedence than + and –
    
    \[
    \text{int } x = 3 + 5 \times 6; \quad \text{int } y = (3 + 5) \times 6;
    \]
  - Parentheses are free, use them liberally

- Arithmetic expressions are evaluated left-to-right in the absence of parentheses
  
  \[
  \text{int } x = 3 \times 4 / 6 \times 2; \quad \text{int } y = (3 \times 4) / (6 \times 2);
  \]

- There are limits on int and double value, be aware of them.

Numeric Input

- Use IntFields to read in numeric data
- Use the getInt method of the IntField class
  - After creating an IntField object we can use method
    
    \[
    \text{object.method();}
    \]
  - Syntax (we’ve used it before) is
    
    \[
    \text{object.method();}
    \]
  - For example note following code fragment
    
    ```java
    // declare and create IntField intin
    IntField intin = new IntField(20);
    // declare n (new not needed)
    int n;
    // n reads value from IntField intin
    n = intin.getInt();
    ```

Game Example with Integer Input

```java
public class AboveBelow extends java.applet.Applet
    implements ActionListener
{
    TextField m1, m2;
    IntField i1;
    Button b1, b2;
    int secret, guess;
    public void init() {
        m1 = new TextField(80);
        m1.setText("Enter number between 0 and 100 below, then push SECRET");
        i1 = new IntField(40);
        m2 = new TextField(80);
        b1 = new Button("SECRET");
        b2 = new Button("GUESS");
        add(m1); add(b1); add(i1); add(b2); add(m2);
        b1.addActionListener(this);
        b2.addActionListener(this);
    }
    public void actionPerformed(ActionEvent event) {
        Object cause = event.getSource();
        if (cause == b1) {
            secret = i1.getInt();
            i1.setInt();
            m1.setText("Now, enter your guess below, then press GUESS");
        }
        if (cause == b2) {
            guess = i1.getInt();
            if (guess == secret)
                m2.setText("You've got it!");
            if (guess < secret)
                m2.setText("The number is greater than "+guess);
        }
    }
}
```

Game Example (page 2)
Game Example continued:

```java
if (guess > secret)
{
    i1.setInt();
    m2.setText("The number is less than "+guess);
}
```

- What is best strategy to play this game?
  - Where have we seen it before?

Types for Numbers

- The type `String` is not a built-in type, actually it’s a class
- There are many numerical types in Java. We’ll use two
  - `int`, represents integers: {...-3,-2,-1,0,1,2,3,...}
    - Conceptually there are an infinite number of integers, but the range is limited to \([-2^{31}, 2^{31}-1]\) or \([\text{Integer.MIN_VALUE}, \text{Integer.MAX_VALUE}]\)
    - Alternatives? Why is range limited?
  - `double`, represents real numbers like \(\pi, \sqrt{2}\)
    - Not represented exactly, so expressions like 100*0.1 may yield unexpected results
    - Double precision floating point numbers, another type `float` exists, but it’s a terrible choice (can generate poor results)

GIGO: program as good as its data?

- In calculations involving floating point numbers it’s easy to generate errors because of accumulated approximations:
  - What is \(10^{23} + 1\)?
  - When is \((x + y) + z\) different from \(x + (y + z)\)?
- The type `int` is severely constrained on 16-bit computers, e.g., running DOS, largest value is 32,767 \(2^{15}-1\)
  - Even on 32-bit machines, how many seconds in a millennium? \(60*60*24*365*1000\), problems?
  - On UNIX machines time is measure in seconds since 1970, problems?
  - What was Y2K all about?

What arithmetic operations exist?

- Syntax and semantics for arithmetic operations
  - Addition, subtraction: `+` and `-`, `int` and `double`
    ```java
    23 + 4  x + y  d - 14.0 + 23
    ```
  - Multiplication: `*`, `int` and `double`
    ```java
    23 * 4  y * 3.0  d * 23.1 * 4
    ```
  - Division: `/`, different for `int` and `double`
    ```java
    21 / 4  21 / 4.0  x / y
    ```
  - Modulus: `%`, (think of it as `remainder`) only for `int`
    ```java
    21 % 4  17 % 2  x % y
    ```
- Mixed type expressions are converted to “higher” type
- Associativity of operators determines left-to-right behavior
- Use parentheses liberally
  - Without () use operator precedence: `*`, `/`, `%`, before `+`, `-`