CSCI 161
Introduction to Computer Science
Last Time...

- We saw:
  - How fields are declared
  - How to define methods and constructors
  - How to call (or re-use) other methods inside algorithms we write

- Still not sure about:
  - *When* to use local variables over fields?
  - When to *return* a value from a method
  - How to properly comment (document) your code
  - How to make "branching" decisions in code?
  - (Address these issues in this lecture)
Ticket Machines as Software

- Ticket machines can be found in most subway and train stations

- What all TicketMachines to:
  - Have a set price/cost for a ticket
  - Print a ticket after user inserts correct money
  - Keep a running total of money collected
  - Assume: Only 1 ticket price and price is in whole dollars (too lazy to count cents)
TicketMachine Demo

- Readings
  - BlueJ book Chap 2

- Let's first take a look at a demonstration of how we expect the TicketMachine to behave

- Project is provided to you on course page
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to Screen
    - If-Statements
    - Local Variables
- Conclusion
We'll name the class TicketMachine.

What *fields* should *all* ticket machines have?

- What do they need to remember about themselves?
  - Amount of money inserted so far (we'll call that the *balance*)
  - Amount of money accumulated over time by the machine (called *total*)
  - Price per ticket (called *price*)

- *What are their data types? (We said we were dealing with dollars in whole numbers)*
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to Screen
    - If-Statements
    - Local Variables

- Conclusion
 Constructors are responsible for setting up the ticket machine's state as soon as it's created

- Recall the syntax

```java
public ClassName(parameterList) {
    // code to initialize object's state
}
```

- Remember, parameter lists might be empty

- One Issue: Well... the balance and total should be zeroed out initially, but what about price per ticket?
Default ("No-Args") Constructor

- **Default constructor**: a constructor that accepts no inputs
  - Also called the "no-arguments" (no-args) constructor

- **Know this**: Java automatically inserts a default constructor if you don't define it.
  - But it might not do what you want!
  - Better to be explicit: Write another constructor that takes no input parameters.
  - *In what initial state should it put newly-created TicketMachine objects?*
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to Screen
    - If-Statements
    - Getters and Setters
    - Local Variables

- Conclusion
What *behaviors* should our ticket machines have?

- Let a user `insertMoney`
- Retrieve the cost of a ticket: `getPrice`
- Retrieve the current balance: `getBalance`
- Let a user `printTicket`. It should also update total and clear the balance.
Ticket Machine (Behaviors)

- For each method, ask:
  - Does the algorithm require any input? (Input params)
  - Does the algorithm owe the caller any values before it terminates? (Return type)
  - That's enough information for us to write the signature.
    - *Always write the method's signature first!!!*

- Recall the **template** for writing methods:

  ```java
  public returnType methodName(ParameterList) {
    statement 1;
    statement 2;
    ...
  }
  ```
  
  **Signature**
  
  **Body**
  (contains the algorithm)
What Should printTicket() Do?

- **Step 1:** We want it to print the following to the screen:

```plaintext
#########################################
# The Puget Sound Line
# Ticket
# 5 dollars.
#########################################
```

This number must reflect the cost of a single ticket at the particular machine
*(Hey, we have a field remembering that value)*

- **Step 2:** After printing, it should clear update the total and clear the balance.
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to Screen
    - If-Statements
    - Local Variables

- Conclusion
How to Print Something to the Screen?

- **Syntax:**
  ```java
  System.out.println(thingWeWantPrinted);
  ```

  By the way, `thingWeWantPrinted` could also be a variable that's storing a String.

- **Examples:**
  ```java
  System.out.println("Hello World!");
  > Hello World    <------ This is what appears on the terminal!
  
  String str = "Hello World!";
  System.out.println(str);
  > Hello World    <------ This is what appears on the terminal!
  
  String str = "Hello\n\tWorld!";  // Note: "\n" means new line, "\t" means tab
  System.out.println(str);
  > Hello
  >     World
  ```
Important: Concatenating Strings

- **Concatenate** is a fancy way of saying, "to append" Strings
  - We can append Strings to other Strings, math expressions, variables, etc.
  - The concatenation operator is the "plus" symbol: +

**Example:**

```java
int x = 100;
System.out.println("The value of x is "+ x);
> The value of x is 100
```

**Another:**

```java
int x = 13 * 4 + 9;
System.out.println("13 * 4 + 9 is\n:"+x);
> 13 * 4 + 9 is
> 61
```
Important: Concatenating Strings! (Cont.)

- Accumulating concatenation symbol `+=` can also be used!

- Example: Build a String variable, then print it out!

```java
String str = "University\n";
str += "of\n";
str += "Puget Sound";
System.out.println(str);
```

```
University
of
Puget Sound
```

```
BlueJ
Terminal
Window
```
Finally... We Can Finish `printTicket()`

- The `printTicket()` method:
  - Step 1: We want it to print the following to the screen:
    
    ```
    # The Puget Sound Line
    # Ticket
    # 5 dollars.
    # This must reflect the cost of a single ticket *(hey, we have a field for that!)*
    ```
  - Step 2: After printing, it should update the total and the balance.
Outline

- Writing Our First Class: TicketMachine
  - Designing Fields
  - Designing Constructors
    - Parameters
  - Designing Methods
    - Printing to the Screen
    - Local Variables
    - If-Statements

- Conclusion
Accessor Methods (Getters) are used to observe the object's current state

- They always return some value
- They have a return type that is non-void
  - Really important (and worth repeating): returning is not the same as printing!

Let's take a closer look at `getPrice()` and `getBalance()`

```java
public int getPrice() {
    return this.price;
}

public int getBalance() {
    return this.balance;
}
```
The Thing about return Statements

- **Important:** When a `return` statement is reached, any successive statements will not be executed!

1. Which method is being called? Execute it!

2. Return statement reached, exit the method and return any value AND control to the caller!

3. Control returns back here! x now has 80. Carry on with remaining code.

```java
public int fizz()
{
    int x;
    x = bizz(30, 50);
    ... more code ...
}

public int bizz(int x, int y)
{
    System.out.println("foo");
    return (x + y);
    System.out.println("bar");
} //dead code
```
Mutator Methods (Setters) are used to change an object's state

- They usually have a `void` return type
  - Just tweaking the object's current state;
  - Usually don't need to `return` any values

- `insertMoney()` is an exception
  - It's a setter that happens to return a value back to the caller

```java
public int insertMoney(int amount) {
    if (amount < 0) {
        System.out.println("Error: negative amount");
    } else {
        this.balance += this.amount;  // set the new balance here
    }
    return this.balance;
}
```
A New Method: Refund

- People have been requesting that our TicketMachine handle refunds!

- Think about what it needs to look like:
  - We can call the method `refundBalance()`
  - It returns the current balance
  - Then it clears the balance to zero

- What's wrong with the following code?

```java
public int refundBalance() {
    return this.balance; //return current balance to user
    this.balance = 0;    //clear the balance
}
```
Local Variables

- We need temporary storage to hold the `balance` before resetting the `balance` field.

```java
public int refundBalance()
{
    int amountToRefund;
    amountToRefund = this.balance;
    this.balance = 0;  //clear the balance
    return amountToRefund;  //return the amount to refund to user
}
```

- Methods and constructors can have their own "local variables"
  - Like input parameters, they only exist between the `{...}
  - Unlike input parameters, the body must set their values
  - They are only accessible within the method (or constructor)
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
      - If-Statements
- Conclusion
Problem!

- **Problem**: Our TicketMachine has been losing tons of money and we don't know why!!!

- Someone points out that our `printTicket()` method is broken... it allows you to print a ticket no matter how much $ you've put in.

- Instead, we want it to make a choice
  - Is the given balance enough to purchase a ticket?
    - If so, print a ticket
    - If not, print the amount that still needs entered
Fixing printTicket()

- Currently... the `printTicket()` code looks like this:

```java
public void printTicket() {
    System.out.println("######################");
    System.out.println("# The Puget Sound Line");
    System.out.println("# Ticket");
    System.out.println("# " + this.price + " dollars.");
    System.out.println("######################");

    this.total += this.price;
    this.balance = 0;
}
```

- What changes need to be made to this method?
  - Is the current balance enough to buy a ticket?
    - If so, print a ticket, accumulate the total, clear the balance just like before
    - If not, print the amount that still needs to be inserted
Conditional Statements

- What if we need to make a decision in our code?

- This is known as an If-Then-Else clause:
  - The `else` clause is entirely optional, but is needed in this case.
  - If-else syntax:

```java
if (some_boolean_condition_is_true) {
    // statements to execute if
    // the condition was true
} else {
    // statements to execute
    // if condition was false
}
```

*What is a boolean condition?*
What are Boolean Conditions?

- Commonly, they are comparisons that result in a "truth value"

- **Comparison operators**
  - 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>(a == b)</td>
<td>Equals?</td>
<td>(x == (93-60))</td>
<td>true</td>
</tr>
<tr>
<td>(a != b)</td>
<td>Not equals?</td>
<td>(4/5 != 1)</td>
<td>true</td>
</tr>
<tr>
<td>(a &lt;= b)</td>
<td>Less than equals?</td>
<td>(3 &lt;= 2)</td>
<td>false</td>
</tr>
<tr>
<td>(a &gt;= b)</td>
<td>Greater than equals?</td>
<td>(3 &gt;= 2)</td>
<td>true</td>
</tr>
<tr>
<td>(a &lt; b)</td>
<td>Less than?</td>
<td>(-(13 * 8) &lt; -10)</td>
<td>true</td>
</tr>
<tr>
<td>(a &gt; b)</td>
<td>Greater than?</td>
<td>(2 &gt; 13)</td>
<td>false</td>
</tr>
</tbody>
</table>
public void printTicket() {
    if (balance >= price) {
        System.out.println("############################");
        System.out.println("# The Puget Sound Line");
        System.out.println("# Ticket");
        System.out.println("# " + this.price + " dollars.");
        System.out.println("############################");
        this.balance -= this.price;
        this.total += this.price;
    } else { //they must not have entered enough
        System.out.println("You still owe " + (this.price - this.balance) + " dollars!");
    }
}
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
  - Using TicketMachines in Code (no more point and click!)
- Conclusion
Scenario

- Let's simulate a real-world scenario:
  - A user
Outline

- Writing Our First Class: TicketMachine
  - Fields
  - Constructors
    - Parameters
  - Methods
    - Printing to screen
    - Local Variables
    - If-Statements
    - More practice with if-then-else

- Conclusion
We'll do the first one together.

What's the output when...
- num1 = 5, num2 = 4

Try these on your own:
- num1 = 5, num2 = 12
- num1 = 5, num2 = 27

```java
public void practice(int num1, int num2) {
    if (num1 >= num2) {
        System.out.println(" red ");
        System.out.println(" orange ");
    }
    if (((num1 + 5) >= num2)) {
        System.out.println(" white ");
    } else {
        if (((num1 + 10) >= num2)) {
            System.out.println(" black ");
            System.out.println(" blue ");
        } else {
            System.out.println(" yellow ");
        }
    }
    System.out.println(" green ");
}
```
Your Turn! Largest of Two

- Write a method:
  - `max` that inputs 2 integers and *returns* the larger one.

```java
public _____ max(_____________________) {

// Your turn!
}
```
Practice (Code Writing)

- Use nested if-else statements

- Write a method called `weather()` that does not return, and inputs an integer called `temperature`:
  - Print "It is cool" when `temperature` is <= 50.
  - Print "It is pleasant" when the `temperature` is between 51 and 80.
  - Print "It is hot" when the `temperature` is between 81 and 95.
  - Print "It is blazing" on one line, and "dress coolly" on the next line, when `temperature` is greater than 95.
Yuck... kind of hard to read with so much "nesting"...
Else-if statements can simplify the nested chain of if-statements!

- They must follow an if or another else-if statement.

```java
public void weather(int temperature) {
    if (temperature <= 50) {
        System.out.println("It is cool");
    } else if (temperature <= 80) {
        System.out.println("It is pleasant.");
    } else if (temperature <= 95) {
        System.out.println("It is hot!");
    } else {
        System.out.println("It is warm. Dress coolly.");
    }
}
```
Consider the code to the left. What values of \( z \) would produce the following results:

1. X
2. XX
3. XXX
4. XXXX
5. No output
Consider the code to the left. What values of z would produce the following results:

1. X --- z < 10
2. XX --- can't be produced
3. XXX --- z > 15
4. XXXX --- z is 10
5. No output --- z is in [11, 15]
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?)

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

WRONG

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

CORRECT

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

WRONG

```java
if (x > 0 && 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

» Writing Our First Class: TicketMachine
  • Fields
  • Constructors
    - Parameters
  • Methods
    - Printing to Screen
    - Local Variables
    - If-Statements

» Conclusion
In Conclusion...

- New concepts:
  - Fields vs. parameters vs. local variables
    - Scope and lifetime
  - Methods: signature and body
    - Role of getters vs. setters
  - Printing to screen with: `System.out.println()`
  - If-then-Else Statement

- Next time: Defining New Classes & More on Conditionals (Chap 3 & 5)
Talk tonight!
- Bias in Algorithms and the Misuse of Big Data Sets
  - Henry Walker, Visiting Professor
  - TH 395, 4pm
  - Refreshments served

Reminders
- Lecture 2 Notes Posted
- Hwk 1 due tonight -- questions?
- Hwk 2 posted, due next Fri

Let's review the syntax templates: class, fields, constructors, methods
Lab 3 tomorrow. Proper lab etiquette

- Work together, not in isolation! Talk!
- Get here on time; read (don't skim) the lab
- Get off your phones; don't do work for other courses <-- Unless you're done with lab

Last time... New concepts:

- Returning values vs printing values
- Strings and concatenation

Today:

- Javadocs commenting
- Accumulation operators (+=, -=)
- if-then(-else) statements
Hwk 2 due tonight
  • Status? Questions?

Lab 4 tomorrow: All about if-then-else statements
  • Changes to Section CA Lab

Where were we before the snow came...?
  • TicketMachine class (take out your notes)
  • Writing Methods:
    - How to design methods
    - We talked about return types, and the return statement
    - We saw how to use if-then-else statements for insertMoney
Lab 3 postmortem

- Good work overall
- Let's check out the solution... (handout)

Important: Using the random number generator

- How would I generate...
  - ... an integer between 0 and 15 (inclusive)?
  - ... an integer between 5 and 49 (inclusive)?
  - ... an integer between -10 and 10 (inclusive)?
  - ... an integer between -20 and -5 (inclusive)?
Last time...

- Importance of JavaDocs commenting
- Representing line-feeds and tabs in Strings
- if-then(-else) statements

Today...

- More practice with if-then-else
- Accumulation operators (+=, -=)
- Defensive programming
- Terminating behavior of return
Hwk 3 to be posted later today
  • TicketMachine lecture too!

Lab 4 post mortem:
  • Random number generation (and manipulation)
  • Math.abs() is useful!
  • Nesting if-then-else statements is a pain
    - Tracking lots of curly braces!!
    - Remember to auto-layout
Hwk 2 due Friday

Last time...

- Reading if-then-else statements
- Returning also terminates method execution

Today

- else-if statements
- More practice with if-then-else
- Automating algorithm execution (no more point and click)