Game-based Teaching Materials

To Teach Java to Middle School Students

Contents

[Introductory Concepts 3](#_Toc457833706)

[Computer Science: What is it? 3](#_Toc457833707)

[Algorithms: The Swap Puzzle 4](#_Toc457833708)

[BuildGame and UpdateGame 5](#_Toc457833709)

[BuildGame and UpdateGame Extensions 6](#_Toc457833710)

[The Intelligent Piece of Paper 7](#_Toc457833711)

[Game Mechanics: Win State 12](#_Toc457833712)

[Variables 13](#_Toc457833713)

[Variables: Box Variables 13](#_Toc457833714)

[Variables to Track Info: Variable Tag 14](#_Toc457833715)

[Variable Types: Int and String 16](#_Toc457833716)

[Instance of an Int 17](#_Toc457833717)

[Methods 18](#_Toc457833718)

[Method Mad Libs: Introduction 18](#_Toc457833719)

[Method Mad Libs: Parameters 19](#_Toc457833720)

[Writing Strings to Console: Intro to Programming 22](#_Toc457833721)

[[Lab 1] Method: What Does a Method Look Like? 24](#_Toc457833722)

[Parameters 26](#_Toc457833723)

[[Lab 2] Drawing Tiles with Parameters 26](#_Toc457833724)

[Conditionals 28](#_Toc457833725)

[If Statements and Flow Control: The Imp Computer 28](#_Toc457833726)

[If/Else Intro Game 29](#_Toc457833727)

[If/Else Intro Game Extension: Else If 30](#_Toc457833728)

[[Lab 3] If/Else Lab 31](#_Toc457833729)

[If/Else Lab Extension: Using Variable Declarations 33](#_Toc457833730)

[[Lab 4] If/Else Lab: Maze Game 34](#_Toc457833731)

[[Lab 5] If/Else Lab: Personalization 36](#_Toc457833732)

[For Loops 38](#_Toc457833733)

[For Loops: Introduction 38](#_Toc457833734)

[[Lab 6] For Loops Lab: Drawing Columns 40](#_Toc457833735)

[[Lab 7] For Loops Lab: Make a Board! 43](#_Toc457833736)

[Gameplay 45](#_Toc457833737)

[[Lab 8] New Methods Lab: Laser and Matching Tiles 45](#_Toc457833738)

[[Lab 9] Gameplay Lab: Win Conditions 47](#_Toc457833739)

[[Lab 10] Gameplay Lab: Cleanup the game! 49](#_Toc457833740)

[Final Lab: Make a Game or 2! 51](#_Toc457833741)

[Sample Camp Schedule: Week 1 52](#_Toc457833742)

[Sample Camp Schedule: Week 2 53](#_Toc457833743)

# Introductory Concepts

## Computer Science: What is it?

**Discussion (non-computer), 15 Minutes**

Concepts:

The definition of computer science and different occupations associated with it.

Learning Objectives:

Students will be able to:

Define computer science and list multiple occupations associated with it.

How to Assess (optional):

Ask students probing questions, such as:

1. What is computer science?

2. What are some careers in computer science?

Materials:

* White board
* Dry Erase Markers

Process:

1. Write the definition on the board
   * + The study of computers and their application
   1. Start a discussion. As the discussion progresses, write students’ ideas on the board.
      * What do you think this definition means?
      * Does this include phones? Calculators? What else does it include?
   2. Continue discussion. As the discussion progresses, write students’ ideas on the board.
      * What kinds of jobs can you get if you study computer science?
      * What kinds of things will you study in computer science?
        + Does anybody already know any languages?
   3. End discussion by talking about how they will be learning a programming language called java by making a game.

## Algorithms: The Swap Puzzle

**Activity (non-computer), 60 minutes**

*Adapted from Queen Mary: University of London*

Concepts:

Algorithms and Testing

Learning Objectives:

Students will be able to:

Demonstrate understanding of an algorithm by coming up with a solution to puzzles presented.

How to Assess (optional):

Observe student solutions

Materials:

* Game board sheets (print from website)
* Game pieces
* Instruction sheets and algorithm recording sheets (print from website)
* Sheets to label squares on giant board
* 7 chairs
* 3 red team bibs and 3 blue team bibs (could make with construction paper and yarn)
* Countdown Timer

Process:

1. View instructions on source website: <http://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/the-swap-puzzle-activity/>

## BuildGame and UpdateGame

**Activity (non-computer), 30 Minutes**

Concepts:

The two primary methods in game development, buildGame() and updateGame().

Learning Objectives:

Students will be able to:

Identify and describe the two primary methods in game development.

How to Assess (optional):

Ask students probing questions, such as:

1. What are the two methods you need to write for a game?

2. What is the difference between them?

Materials:

* A ball (or other small object that can be used for your “game”)

Process:

1. Explain to the class that you are going to demonstrate the 2 methods necessary for making a game. The first one is buildGame!
2. Get 5-7 student volunteers and bring them to the front of the room.
3. Position them in one row so they are all facing the audience. Have a little fun with it. Tell each person where to stand. You can step back, look at the group, have some switch places. Give the first person in the row the ball.
4. When you’re done, look at the class and say “I’m done!”
5. Ask them what they think the other method would need to be.
   1. Try to guide the direction so that they start talking about how nothing is happening. None of the people do anything.
6. Explain that the next method is updateGame.
7. Tell the class that the goal of your game is to get the ball all the way to the last person in line.
8. Give each person in line instructions.
   1. Every time updateGame is called, spin in a circle and pass the ball.
   2. Every time updateGame is called, make a machine noise and pass the ball.
   3. Every time updateGame is called, give the next person a hi-5. Every 2nd time you call updateGame, pass the ball.
   4. Every time updateGame is called, give the person before you a hi-5. Every 2nd time you call updateGame, pass the ball
   5. Every time updateGame is called, put your hands in the air. If you have the ball, say “YOU WIN!” when you put your hands in the air.
9. Run the game! The teacher should say “UpdateGame” over and over again until the game is won.
10. Have the students sit down. Start a discussion:
    1. What are the methods we will need to write for our games?
    2. What kind of things go into buildGame?
    3. What kind of things will need to go in updateGame?

## BuildGame and UpdateGame Extensions

1. Add a “bug.” Make it so that no matter how many times you call updateGame, you lose. Have the class fix the bug.
2. Go outside. Split the class into 2-3 teams. Have them each make their own “games.” Have them share their games with each other. Maybe even compete!
3. You can add if/else statements into the updateGame method of the game!
   1. Ex. If you are wearing blue, give the ball to the person after you. If you are not wearing blue, skip the person next to you and give it to the person after that.

## The Intelligent Piece of Paper

**Activity (non-computer), 20 - 30 Minutes**

*Adapted from “The Intelligent Piece of Paper” by Peter McOwan and Paul Curzon of University of London with support from EPSRC and Google.*

[*http://www.cs4fn.org/teachers/activities/intelligentpaper/intelligentpaper.pdf*](http://www.cs4fn.org/teachers/activities/intelligentpaper/intelligentpaper.pdf)

Concepts:

What a computer program is and how everything computers do simply involves following instructions written by (creative) computer programmers.

Learning Objectives:

Students will be able to:

Define the role of the programmer vs the program in computer science.

How to Assess (optional):

Ask students probing questions, such as:

1. Is a computer intelligent?

2. How does a computer know what to do?

Materials:

* White board
* Dry Erase Markers
* Copy of the intelligent piece of paper (possibly laminated)
* (optional) a musical greeting card that plays some appropriately horrible song.

Process (taken directly from the pdf) :

**The grab**: Announce that the piece of paper you are holding is more intelligent than anyone in the room (even the highly intelligent teachers there). Wax lyrical about how intelligent it is without saying why. Wave it around while keeping the written side hidden from the audience.

**The set-up:** Ask the audience if they believe you and have a show of hands: first, those who believe it is intelligent and then those who “believe I am talking total garbage and no way just a piece of paper could be intelligent”.

Usually most will go for the garbage option. Congratulate them on their wisdom (both for believing such a wise person as you and especially those that don’t – after all no good scientist believes claims of random people making great pronouncements, however great they are, without some evidence.

Ask them to bear with you for a while – perhaps it is intelligent, perhaps not, but ask for suggestions of what it might be about the paper that could be the basis of such an outrageous claim.

You may get suggestions such as it is something special about the ink, or that it is laminated. With suggestions like the former, praise them for an interesting idea but ask how that exactly might make it intelligent? Ask if the rest of the audience think it would be enough. If the latter, explain that the plastic covering is not the special thing. It is just there to protect the piece of paper.

Another common suggestion is that there is a computer embedded in the paper. This is an opportunity to bring out a musical greeting card that plays some irritating music (and is intelligent enough to know to do it on your birthday) and explain it works via an embedded chip. You may want to mention that such a chip is as complex as the embedded computer used to put Neil Armstrong on the moon. Note that you could have done it that way with your intelligent piece of paper, but you didn’t.

Another suggestion will be that it is what is written on the paper that makes it intelligent. Ask what might be written that would make paper intelligent. Great equations? Wonderful poetry? Exciting facts? Suggest examples and ask if the audience think that would give the paper intelligence. If not, then we need to look for something more. Talk about the fact that knowledge isn’t the same as intelligence – and that they surely don’t just try to memorise things for exams but try to understand which isn’t the same. Agree that writing such things on the paper wouldn’t be enough for it to be intelligent.

Point out that to convince us that it is intelligent it must be able to do something to show that intelligence. What can the paper do? Well it has never lost a game of noughts and crosses (and it plays regularly against humans). ***Note: Noughts and Crosses is the same as Tic-Tac-Toe. Use the name that is more familiar to you and your students.*** Remind them that the game should end in a draw if both players play perfectly. You cannot force a win.

Despite that, the paper has won about half the games it has played against humans like themselves, and drawn the rest. It is a perfect intelligence. Humans aren’t!

Ask them if they believe you. Would they rather see some evidence? To show the evidence you will need two volunteers.

The activity: Draw a noughts and crosses board on the whiteboard/flip chart. Give each volunteer a pen. Explain that to see how intelligent the paper is you will need to play a game of noughts and crosses. It will not be a battle between two humans but between paper-kind and humankind. The paper is ‘peripherally challenged’ – you didn’t bring it a robotic arm or camera system (computer peripherals) so it needs a servant to do its bidding. You might want to note that just because someone is paralysed doesn’t mean they aren’t more intelligent than you.

One person will therefore play for the piece of paper. Their job is to just do what they are told by the paper. They must switch off their highly intelligent brain and do exactly as they are commanded: we don’t want to know how well they play the game, just how well the paper plays. They should just read out loud the paper’s instructions (so everyone can tell it is the paper playing not them) and do as it says.

The other player is there to represent the best of humanity. It may be best not to pick someone who was really keen and appearing to think they would never lose to do this to reduce the chances of it being a drawn game (though often such volunteers still lose). Their job is to use all their intelligence to play as well as they can. As the paper is so intelligent, to make it fairer, say they can get help from the audience if they need it. Tell the audience to shout out if they think a mistake is being made or know the move to make.

Now get the person playing for the paper to read its instruction starting with it wanting to go first. Comment that it is quite clever of it to want to go first. If someone complains about it being unfair, point out that the game should just end in a draw whoever goes first. Going second isn’t a reason to lose.

The paper’s servant should then read out the first move and make the move: playing in a corner. Over to the human. There may be lots of shouts about different places to go. If the person is unsure encourage them to go where they think best of the options shouted. Continue like this, making sure the reader does read out and follow exactly the instruction and helping them understand where they are being told to go if need be. For example, “opposite corner”, means the diagonally opposite corner. This can later lead to a discussion of why special programming languages are needed: to be precise about what is to be done.

Sometimes after the second or third move people in the audience will declare the game lost. Point out humanity does often resign at this point. Remind them it is only a piece of paper though. It might just have been lucky this far so might still mess up and not see what they can see.

If the paper can win two ways, then often the player will jokingly cheat, such as drawing two 0s. Point out that humanity often resorts to cheating at this point, and make them play properly, reminding them it is only paper and may still go wrong.

Either the paper will win or it will be a draw (if the human realises they need to go on the side not in the corners). If the latter then remind everyone that that was predicted but that even so it was still clever of it to not let the human win. Announce that once more it has kept its unbeaten run. Say you’ll accept that perhaps it’s not more intelligent than the humans but it has shown itself to be their equal. If the paper wins announce that yet again the paper has shown itself more intelligent than humanity and that you have shown them the evidence asked for.

Either way ask for applause both for the volunteers as they return to their seats and the paper for its stunning performance.

The explanation: Point out they asked for evidence and you have given it. Ask again for a show of hands as to who now believes the paper is intelligent and who believes you are talking garbage and paper can’t be intelligent. Usually everyone now is sure it isn’t intelligent, despite the evidence of its abilities.

Point out that it did show intelligent behaviour so there is intelligence somewhere.

Where is it? Someone will almost certainly say it is in the person that wrote the instructions. Ask if everyone agrees that that is where the intelligence is and get a show of hands. (You can then thank them from me for recognising my great intelligence :-) )

Now explain that what is on the paper is essentially a computer program: instructions to be blindly followed. Everything they have ever seen a computer do, it was just doing by blindly following instructions in the same way. Point out that if they are saying that the paper is not intelligent because it is just following rules, then they are saying no computer could ever be intelligent either.

These instructions were written in a language so that a human can follow them. If they were to be written for a computer they would be written in a programming language: just a language a computer can understand precisely and so follow. Point out that if they think that it is the writer of the instructions who is the creative intelligent one, then they are saying that computer programmers are intelligent and creative (which is true). It is computer programmers who have written all the instructions all those computers are following.

The intelligent piece of paper is on the next page. Have fun!

I am a highly intelligent piece of paper.

Let’s play Tic-Tac-Toe.

I am X, and I go first.

**Move 1**:

Go in a corner.

**Move 2**:

IF the other player did not go there

THEN go in the opposite corner to move 1.

ELSE go in a free corner.

**Move 3**:

IF there are 2 Xs and a space in a line

THEN go in that space.

ELSE IF there are 2 Os and a space in a line THEN go in that space.

ELSE go in a free corner.

**Move 4:**

IF there are 2 Xs and a space in a line

THEN go in that space.

ELSE IF there are 2 Os and a space in a line THEN go in that space.

ELSE go in a free corner.

**Move 5**: Go in the free space

## Game Mechanics: Win State

**Discussion (non-computer), 45-60 Minutes**

Concepts:

Implementing a win state

Learning Objectives:

Students will be able to:

Implement a win state in the game

How to Assess (optional):

Observe student work

Ask students probing questions, such as:

1. How do you know if you win the game?

2. What happens when you win the game?

Materials:

* Computer with BlueJ and CorruptedFunctionalAPI installed (for each student)
* Projector showing teacher screen

Process:

1. Open the MyGame class (make sure it’s empty of any previous work).
2. Write a series of drawTile() statements (4-5) in buildGame().
3. Now, tell the students you want to create a game, but this one will have a WIN CONDITION.
   1. Ask: What is a win condition?
4. Walk through the steps of recreating the program which will draw tiles based on pressing right, left, up, and down. Walking through this may feel slow, but it is important to review/reinforce these concepts before moving forward. Take your time! It’s an important part of the lesson.
5. Declare the variable int numberOfTiles = 0 at the top.
   1. Every time a tile is drawn, add one to that number.
6. At the end of the updateGame method, include an if statement that checks for a win condition.
   1. For example, if (numberOfTiles == 20) { gameWin(); }
7. Have students implement a win scenario for the game on their computer. They can edit the game or change the win condition however they want.
8. Give them time to play each other’s games.
9. End with a discussion:
   1. What win conditions were fun to play?
   2. What made them fun? Where they very hard? Very easy?
   3. Did you get ideas from seeing other people’s games? Would you change the win condition for your game after observing your classmates’ games?

# Variables

## Variables: Box Variables

**Activity (non-computer), 60 minutes (USE EXTENSION EXERCISES)**

*Adapted from Queen Mary: University of London*

Concepts:

Variables and Assignment

Learning Objectives:

Students will be able to:

Demonstrate understanding of variables and assignment.

How to Assess (optional):

Observe student solutions

Materials:

* 3 boxes (shoeboxes would probably work)
* Variable name labels (print from website)
* Value cards
* Program slides (obtain from website)

Process:

1. View instructions on source website: <http://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/the-box-variable-activity/>
2. Do the introductory lesson as well as the Assignment Dry Run Activity on the website.
   1. <https://teachinglondoncomputing.files.wordpress.com/2014/05/activity-assignmentdryrunjava.pdf>
   2. Don’t just give the students worksheets! Do the problems together on the board or using the physical boxes.

## Variables to Track Info: Variable Tag

**Activity (non-computer), 60 min (or however long you want)**

Concepts:

Using variables to keep track of information.

Learning Objectives:

Students will be able to:

Demonstrate an understanding of initializing a variable and incrementing it.

How to Assess (optional):

Ask probing questions, such as “What information did you have to keep track of?” and “What would be a good variable name for this?”

Materials:

* A space outdoors to play tag
* A whiteboard and markers for discussion at end (in classroom).

Process:

1. Have students go outside. Tell them they are going to play tag.
   1. You can use any type of tag game you like. For this example, I will use Sharks and Minnows.
2. Have a volunteer (or 2) be the sharks. They should stand in the center of the field. Everyone else should line up along one end of the field.
   1. The rules:
      1. When the sharks yell “High Tide!” all of the minnows have to run to the other side of the field. Once they reach the other side of the field, they are safe.
      2. No minnows can run off the field. They can only run on the space in the designated field.
      3. Once a minnow has been tagged, it is frozen in place. It is now a frozen shark. It can’t move from the spot where it froze, but it can tag any minnows that run past.
      4. The sharks cannot yell “High Tide” again until all untagged minnows are safely across the field.
      5. The game ends when everyone is a shark. The last minnow to be tagged is the first shark for the next round.
3. Have students play one regular version of the game.
4. The second time, tell them that each minnow must be tagged 3 times before they are frozen. The sharks cannot tag the same minnow twice in a row.
   1. Each minnow needs to keep track of how many times they have been tagged (honor system!).
5. Let them play with this new rule.
6. After playing, go inside for a discussion.
7. On the board, write buildGame() and updateGame(). Have the students brainstorm what would go in each method for the tag game they just played.
   1. When they bring up tracking how many times they were tagged, tell them that they would need to declare an int to keep track. Have them name it and write some pseudo code to show how it would work
      1. Ex. int tagCount = 0;

if( minnowIsTagged() == true)

{ tagCount = tagCount +1; }

1. Discuss:
   1. What other things might you need to keep track of in a game? Think of your favorite games you like to play!
      1. Examples: health, money, points, etc
   2. What might we keep track of with our tile games?

## Variable Types: Int and String

**Lesson 15 Minutes**

Concepts:

Variable types: int, String

Learning Objectives:

Campers will be able to:

Discriminate between the different kinds of variables

How to Assess (optional):

Ask student to identify an int, String, and boolean in their code

Materials:

* White board and markers

Process:

1. Discussion: INTRO
   1. Review of the concept that computers have their own language
   2. There are some things we want to be able to communicate with the computer.
2. Discussion: INTS
   1. Let’s talk about how to communicate whole numbers!
   2. They are integers, so we use the keyword “int”
   3. In a game, what are some examples of int we might want to use?
3. Discussion: STRINGS
   1. Let’s talk about how to communicate words!
   2. Any words, sentences, etc are stored as Strings.
   3. Remember—the word ‘String’ is always capitalized
   4. Strings HAVE TO be in quotation marks
   5. In a game, what are some examples of Strings we might want to use?

## Instance of an Int

**Lesson 30 Minutes**

Concepts: Creating an instance of an int.

Learning Objectives:

Campers will be able to: Declare an instance of an int

How to Assess (optional):

Observe student has declared an int in their code.

Materials:

1. 1 stapler, 1 piece of paper, 2 erasers
2. 4 index cards with names on them: Bob, Popeye, Flounder, Gracie

* Tape on the back so they can be attached to items.

1. White board and markers

Process:

1. Show class the stapler, piece of paper, and two erasers.
2. “Name” each of those items using the index cards.
3. Discussion:
   1. Has the name changed the item?
   2. Do they do different things? Or do they still perform the same functions as before?
   3. Do I need to say, “The stapler, Bob” or can I just say “Bob”?  Do you know what I’m talking about if I just say “Bob”?
   4. Let’s look at the erasers.  Let’s say I give it a command like “Flounder, erase the board.” (proceed to erase the board with Flounder).  Do all the erasers erase the board?  How did you know which one should be used?
4. Write on the board:
   1. int flower = 5;
   2. int banana = 3;
   3. int zebra = flower + banana;
5. Discussion:
   1. What does zebra equal?  How do you know?
   2. Why do we name things?
   3. Notice that I didn’t write “int zebra = int flower + int banana”.
      1. You only have to tell the computer it is an int (or a string) once. It will remember!
   4. Consider this:  int health = 100; How would we use that in a game?
6. Have students open BlueJ and practice declaring ints and performing math. They can check their results using System.out.println();;

*Extension (optional): Declare and name Strings.*

# Methods

## Method Mad Libs: Introduction

**Game (non-computer), 30 Minutes**

Concepts:

The programmer can call methods, which each have their own instructions.

Learning Objectives:

Students will be able to:

Demonstrate knowledge of the concept of method calls.

How to Assess (optional):

Ask students probing questions, such as:

1. Does the main method have to have all the code for the whole program in it?

*(A: No, there can be other methods)*

2. How does the main method access other code?

*(A: By making a method call OR by calling the name of the method)*

Materials:

* Set of mad lib index cards (see attached page for specifications).
  1. Cards should be added/omitted as needed.
* A mad lib
  1. The mad lib should be written on the board or on a PowerPoint slide in the following format:
     + All story text should be written as: System.out.println(“ mad lib text”).
     + All blanks should be written as method calls.
       - getAdjective();
       - getAdverb();
       - getVerb();
       - getNoun();
       - a method for any other blanks that are in the mad lib

Process: *note: this game can be played indoors or outdoors*

1. Get student volunteers for each method. Give each volunteer a card
2. The students with cards should stand on the opposite side of the room from the teacher.
3. The teacher explains to the class that they (the teacher) are the computer and, as the computer, they are going to process the main method.
4. Point to the first line and read it. As soon as you get to a method call. Stop.
5. Start a discussion!
   1. Ask the students what that means? Where is the code if it’s not in the method? Ask if any of them knows what it means? Does anyone have a card with that? (If student with that card doesn’t respond, prompt the student volunteers to read their cards).
6. As the ‘computer’ the teacher, should go to the other side of the room and ask the student with the appropriate method to read their card. Then, return to read the method in the front of the room.
7. Insert the word the student gave you and continue the story.
8. When you get to the next method call, repeat questions you asked after the first card. Students should respond faster.
9. Continue with this until all the way through the madlib.
10. After this, bring students together for a group discussion. Ask the following questions (or similar question):
    1. What did you learn?
    2. Did the main method know what the other methods did? Did any of the methods know what the other methods did?
    3. Which do you think is easier—writing all the code for getVerb() every time you need a verb, or calling the method getVerb()? Why?
    4. How might you use this in your program?

Card Examples

*Feel free to adapt or create new ones (you can do this activity multiple times!)*

|  |  |
| --- | --- |
| getVerb(); | public String getVerb() {  //You get to pick a past-tense verb to tell the teacher!  return random past-tense verb; } |
| getAdjective(); | public String getAdjective() {  //You get to pick an adjective to tell the teacher!  return random adjective; } |
| getNoun(); | public String getNoun() {  //You get to pick a noun to tell the teacher!  return random noun;} |

## Method Mad Libs: Parameters

**Game (non-computer), 30 Minutes**

Concepts:

Reinforce the concept that the programmer can call methods, which each have their own instructions.

Introduce parameters.

Learning Objectives:

Students will be able to:

Demonstrate understanding of what a parameter is.

How to Assess (optional):

Ask students probing questions, such as:

1. What is a parameter?

*(A: A variable that you provide the method which the method will use)*

2. What is an example of a parameter?

*(A: When we gave getVerb(1) a one, the method returned an -ing verb.)*

Materials:

* Set of mad lib index cards (see attached page for specifications).
  1. Cards should be added/omitted as needed.
* A mad lib
  1. The mad lib should be written on the board or on a PowerPoint slide in the following format:
     + All story text should be written as: System.out.println(“ mad lib text”).
     + All blanks should be written as method calls.
       - getAdjective();
       - getAdverb();
       - getVerb(int v);
       - getNoun(int v);
       - a method for any other blanks that are in the mad lib

Process: *note: this game can be played indoors or outdoors*

1. Get student volunteers for each method. Give each volunteer a card
2. The students with cards should stand on the opposite side of the room from the teacher.
3. The teacher explains to the class that they (the teacher) are the computer and, as the computer, they are going to process the main method.
4. Point to the first line and read it. As soon as you get to a method call. Stop. If it has no parameters, quickly review the concept by asking the class that happens.If it has a parameter, move to step 15.
5. Start a discussion!
   * 1. Ask the students what that means? Why is there a number there? Ask if any of them knows what it means? What method should the computer go to?
6. As the ‘computer’ the teacher, should go to the other side of the room and CLEARLY STATE THE NUMBER (parameter) FOR THE STUDENT WITH ATHAT CARD. The student should read the card and provide an appropriate word. Then, the teacher should return to read the method in the front of the room.
7. Insert the word the student gave you and continue the story.
8. When you get to the next method call, repeat questions you asked after the first card. Students should respond faster.
9. Continue with this until all the way through the madlib.
10. After this, bring students together for a group discussion. Ask the following questions (or similar questions):
    * 1. What did you learn?
      2. What is a parameter?
      3. Which do you think is easier—writing a separate method for each type of verb you might want? Or writing one method that takes in a parameter? Why?
      4. How might you use this in your program?

Alternate/Extension

Add a bug to the code! Write “getBerb()” as part of the program. Talk the students through what the computer will do when it sees that (stop/report error).

Card Examples

|  |  |
| --- | --- |
| getVerb(int v); | public String getVerb(int v) {  if (v is 1)  //You get to pick a verb that ends in -ing to give the teacher!  else if (v is 2)  //You get to pick a past tense verb to give the teacher  Else if (v is not 1 or 2)  return random verb; } |
| getAdjective(); | public String getAdjective() {  //You get to pick an adjective to tell the teacher!  return random adjective; } |
| getNoun(int v); | public String getNoun(int v) {  if (v is 1)  //You get to pick a singular noun to tell the teacher!  else if (v is 2)  //You get to pick a plural noun to tell the teacher!  else if (v is not 1 or 2)  return random noun;} |

## Writing Strings to Console: Intro to Programming

**Computer Activity: 30 Minutes**

Concepts:

How to use the System.out.println() method

Learning Objectives:

Students will be able to:

Implement System.out.println()

How to Assess (optional):

Look at output on student computers.

Materials:

* Computer for each student with Bluej
* Projector showing teacher’s computer screen

Process:

1. Have the students start with their computers closed so their attention is on the projection showing the teacher’s screen.
2. The teacher should talk about how they want to write a sentence to give the computer a command.
3. Ask the students what some examples of commands are. They can think of commands they give their dogs.
   1. They will likely say things like: Sit! Down. Stay!
4. Ask the students what these commands have in common. What are the parts of the command (that all commands need)?
   1. They will likely say things like: verb, name, period, exclamation point
5. Tell them that commands for computers are similar. There are certain things the computer expects to see with a command: A method call (which is like a verb) and a semi-colon (punctuation).
6. Introduce the System.out.println() method. This method gives the computer the COMMAND to print out the String to the console (or screen).
   1. Give an example like:
      1. System.out.println(“My name is Jennifer.”);
7. Explain to them that they are going to practice using this command. They need to call the method at least 5 times to answer the following questions (they can use more lines if they want):
   1. What is your name?
   2. What school do you go to?
   3. What is your favorite subject in school?
   4. What is your favorite game?
   5. What do you want to be when you grow up?
8. Have them open BlueJ on their computers and give them time to play with this command. Walk around the room to support.
9. After the students have successfully done this, bring the class’s attention back to the front of the room for discussion.
   1. What happened if you didn’t put a semi-colon at the end? What if you put a period instead? Try it!
   2. What happens if you forget the quotation marks? Try it!
   3. What did you learn about writing commands for the computer?

## [Lab 1] Method: What Does a Method Look Like?

**Activity, 1 hour**

Concepts:

The format and parts of a method in java.

Learning Objectives:

Students will be able to:

Write a method in java.

How to Assess (optional):

Look at student output to verify their methods compile and are working as desired.

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Open BlueJ and go into the “MyGame” class.
2. Tell the students you are going to write a method. First, talk about what a method is.
   1. A named section of a program that performs a specific task.
3. Tell them the first step in writing a method is to identify the task you want to perform.
   1. You want to build the world for your game.
   2. Write “build world” on the screen
      1. Ask students why there is an error?
      2. Answer: It needs to be written in a language the computer understands!
4. The first word a computer needs when reading a method is “public” or “private”
   1. Public means that methods from other places in the program can read this method.
   2. Private means that only the methods in the current class file can read it.
   3. Tell the students that everything they will write during camp needs to be public.
5. Erase the “build game” and write “public”
6. The next step is to tell the computer if it needs to give you back any information.
   1. For example, when you use a calculator and type in “546 X 332” — do you want the computer to calculate it and just sit there? Or do you want it to tell you the answer?
   2. We aren’t going to need the program to give us any information back, so we will write “void”—which means nothing.
7. Write “void”
8. Now we need to name our method! *Note: You will need to name it based on what it is named in the API. It will either be buildGame().*
9. Tell them it is important to name things in a way that makes sense. Also, explain that programmers use camel casing. The first word has a lower-case letter, the rest have an uppercase letter.
10. Write the name of the method on the screen.
11. Then explain that we need to tell the computer where the instructions will start and end. We do this with curly brackets! If you forget the brackets, the computer will get confused and not know where to look. Often people forget the end bracket so the computer doesn’t know when to stop reading. So, put both brackets on the screen now with some spaces in between to write the instructions. *Note: This is also helpful because BlueJ will create a colored box making it easier for students to identify where to write*
12. To build the world for the game, you need tiles. I am going to give the computer a command. It is a command the computer already knows. It is
    1. drawTile();
13. Write drawTile() and compile. A tile will be drawn on the screen.
14. Now give students time to explore. Leave your method on the screen for reference.
    1. Challenge students to add 5 or 10 tiles
    2. Students will likely try asking how to change colors or put tiles at specific locations. Tell them they will learn those things later. In the meantime, they can put as many tiles on the screen as they want.
15. After students have done this, start a discussion.
    1. What are the parts of a method?
    2. Can you set up the whole game with just the drawTile() method? Why not? What else do we need to do?

# Parameters

## [Lab 2] Drawing Tiles with Parameters

**Lab (computer), 90 minutes – 2 hours total (may want break in middle)**

Concepts:

Using parameters to provide variable information to methods.

Learning Objectives:

Students will be able to:

Use a method that requires parameters

How to Assess (optional):

Observe students using methods with parameters in the lab.

Ask students probing questions, such as:

1. What is a parameter?

2. What parameters are you using? What do they do?

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Discuss the work from the previous lab. It was a series of drawTile() statements. Start a discussion:
   1. Review: What does this code do?
   2. What DOESN’T it do that you want it to do?
2. Give a working definition of parameters
   1. Some methods will let you give them information (in the form of ints and Strings) and it will use that information to complete its task.
3. Tell them to think of the game screen as a grid. Each location can be identified using the (x,y) coordinate system. If students don’t understand it, review it.
   1. Starting at the bottom left corner, x will tell how many places to move right. Y will tell how many places to move up.
4. Demonstrate by drawing a few tiles
   1. drawTile(3,5);
   2. drawTile(10,10);
5. Have students explore. Their task: to figure out the size of the grid! What is the largest x number they can put in before it goes off the screen (should be 25)? What is the largest y number (should be 9)?
6. Group discussion
   1. Have students share the numbers they got for the max values.
7. Demonstrate drawing a few tiles with specific colors
   1. drawTile(3,5, “blue”);
   2. drawTile(10,10, “red”);
8. Have students explore. Their task: draw a picture using the tiles
9. Share pictures & Group discussion
   1. What colors were you able to use?
   2. What colors didn’t work? What happened when you tried to use them? Why do you think this happened?

*Extension (or possibly just a class discussion):*

What if I declared int flower = 3 at the beginning of my program? Could I draw a tile by saying:

drawTile(flower, flower, “blue”);

Try it! Named variables can be used as parameters!

# Conditionals

## If Statements and Flow Control: The Imp Computer

**Activity (non-computer), 90 minutes**

*Adapted from Queen Mary: University of London*

Concepts:

If Statements and Flow Control

Learning Objectives:

Students will be able to:

Demonstrate an understanding of conditionals and flow control by drawing flow charts for a program.

How to Assess (optional):

View student flow charts.

Materials:

* Baton (could be paper towel roll, plastic tube, etc)
* Brightly colored rope/yarn cut into at least 8-meter-long pieces
* Instruction cards (print from website)
* Box (shoebox will work)
* Blank paper
* White board and dry erase markers
* The program (from website)

Process:

1. Instead of imps, it might be better to do monkeys as more students will know what those are.
2. Instructions can be found on source website: <http://teachinglondoncomputing.org/resources/inspiring-unplugged-classroom-activities/the-imp-computer-activity/>
3. Also do extension activity: Drawing Imp Computers
4. After completing the lesson described on the website. Have students draw a flowchart for their tile game.
5. Have a group discussion where students share their flowcharts.
   1. Were they all the same?
   2. Were some different?
   3. If they were different, do they both work?

## If/Else Intro Game

**Game (non-computer), 30-45 minutes**

Concepts:

Introduction to conditional logic

Learning Objectives:

Students will be able to:

Identify if/else as terms the computer understands.

How to Assess (optional):

Ask students probing questions, such as:

1. Where might you see this kind of logic in a game?

Materials:

* Space—can be inside or outside (preferably outdoors)

Process:

1. Have the students line up on one side of the space. The teacher stands on the other.
2. Explain the rules of the game.
   1. The teacher will give an instruction.
   2. Follow the instructions.
   3. The first person to get to the side of the space that the teacher is on is the winner.
   4. Play honestly!
   5. The teacher is the judge! In the case of a tie, they will determine the winner.
3. The teacher should give the first instruction.
   1. “IF you are wearing blue, take one step forward. ELSE jump 3 times.”
   2. Make sure the students follow the instructions. Repeat if necessary.
4. Continue to give instructions in the same format.
   1. Suggestions for the condition include:
      1. Clothing colors
      2. Hair color
      3. Eye Color
      4. Grade Level (likely 6, 7 or 8 for middle school students)
      5. Favorite game
   2. Suggestions for actions include:
      1. Steps (forward, backward, left, right)
      2. Jumps
      3. Turn in a circle
      4. Touch your toes
5. Once someone reaches the end, name them the winner!
6. Allow the game to be played again with the winner giving instructions.
   1. Be prepared to support them to make sure they stick to the IF/ELSE format.
   2. Inform them they are only allowed to use conditionals and actions you used. If they want to use a new one, they must ask you first.
7. After the game is complete and they have returned to their classroom seats, start a discussion.
   1. What information did you need to provide? Support them to come up with the following list (they may use different words, but the concepts should be the same):
      1. The word “If”
      2. A condition
      3. The word “Else”
      4. A command
   2. Where might you use this in a game? Let them brainstorm!

## If/Else Intro Game Extension: Else If

1. Set up the same activity as you did for the If/Else game.
2. Follow the same process.
3. This time, add an ‘else if’ statement.
   1. Example: “IF you’re wearing green, take a step forward. ELSE IF you are wearing blue, take a step to the right. ELSE take a step back.”
4. You will need do this very slowly the first time. After the IF statement is read, stop and let those students move. Then state the ELSE IF. You might notice some students wearing blue (who moved in the first if) moving again because they are also wearing green. Stop them! Explain how the ELSE IF works. Because it says “Else,” and they already met the first condition, they don’t perform the new one. They will only ever perform one of the statements!
5. Continue the game as normal, with the inclusion of some ELSE IF statements.
6. Afterwards, have a discussion in the classroom.
   1. How was this different from the first game?
   2. How might this be used in a game?

## [Lab 3] If/Else Lab

**Lab (computer), 30-60 Minutes (depending on group’s comfort with material)**

Concepts:

Conditionals

Learning Objectives:

Students will be able to:

Demonstrate an understanding of conditionals.

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Review: What are conditionals? If/Else statements! When might we use these?
2. Inform them they will be implementing if/else statements in their code today
3. Begin demonstration of how to write an if/else statement.
   1. Review the concept of if/else statement if necessary.
   2. Start with just an if statement in the updateGame method.
   3. Tell them it always starts with the word if. Then open and close parentheses. The parentheses contain the “what” of the question “if what?” In this case, we want to do something if the user presses the right arrow.
      1. if(pressingRight())
   4. Now they need to make an open curly bracket and a closed curly bracket. (I like to put some space between them to type in.) This is where you put the “do this”. The code reads “if <pressing right>, do <this>.” In this case, we want to draw a tile.
      1. { drawTile(5, 5, “green”); } //you can do this on multiple lines to make it easier to read
   5. Tell them to try writing an if statement. For right now, draw the same tile as you.
   6. Now introduce the else (use the same terminology you used above—curly brackets, etc). In this case, we want to draw a different tile:
      1. If(pressingLeft()) { drawTile(7,7,”red”); }
   7. Have them try running the code!
   8. Now play with it. What kinds of things can they get their program to do?
      1. With the else, try to have them use pressingLeft(), pressingRight(), pressingUp(), and pressingDown(). They should use all 4!
4. Discussion:
   1. What do if/else statements allow us to do?
   2. How were we limited? Answer: only draw at most 4 total tiles

## If/Else Lab Extension: Using Variable Declarations

**Lab (computer), 30-60 Minutes (depending on group’s comfort with material)**

Concepts:

Conditionals and Variable Declarations

Learning Objectives:

Students will be able to:

Demonstrate an understanding of conditionals and variable declarations.

How to Assess (optional):

Observe student work on their computers

Materials:

* Computers with BlueJ and CorruptedFunctionalAPI (for each student)
* Projector showing teacher’s screen

Process:

1. Review: So far, we have worked with if/else statements to draw tiles. What if I want to be able to draw more than 2-4 tiles?
2. I’m going to keep track of the number of times I press right.
   1. At the top, declare int timesPressingRight = 0;
3. Move the cursor to the if section for pressing right. Ask the students, what do we need to do after we press right? Answer: add one to the number!
   1. timesPressingRight = timesPressingRight +1;
4. Now, how can we use this number to draw the tiles in a different location each time?
   1. Let students brainstorm. Lead the discussion to talk about:
      1. We tell the method where to draw it using x and y. Can we alter x and y?
      2. Will it be different each time if we add or subtract the timesPressingRight from one (or both) of those values?
      3. Once they come up with an idea, try it out!
      4. Repeat this until there is a version that draws a new tile in a new spot every time you press right.
5. Let the students explore! Tell them they might try tracking how many times they press left, up, and down as well. This make take some time. Walk around the room and provide support.
6. End with a discussion:
   1. This is a complex topic. Ask if there are any questions. Also, bring up questions that were asked during the lab time which others might find helpful.
   2. What did we add to our if/else? Why did we add it?
   3. What did it help us do?

## [Lab 4] If/Else Lab: Maze Game

**Lab (computer), 90-120 Minutes (depending on group’s comfort with material)**

Concepts:

Conditionals and Variable Declarations

Learning Objectives:

Students will be able to:

Demonstrate an understanding of conditionals and variable declarations.

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer
* Graph paper and pencils for students (optional)

Process:

1. Review: So far, we have worked with if/else statements as well as declaring ints.
   1. What did the if/else statements let us do?
   2. What did we use the ints for?
2. Let’s make a game! First we need to build our world.
3. Choose a player color. You can choose any player color you want. Draw a tile in that color at 0,0.
4. Now, instruct the students that they will use the other tile colors to build a “maze” that the player has to move through. You can use as many tiles as you want to make your maze, but the students should be required to use at least 10.
5. How the game works: the player will get to use the arrow keys to draw a line of tiles through the maze. Don’t cross any borders or you’ll lose!
6. Demonstrate how to write pressing right.
   1. Start by declaring int x = 0 and int y = 0 at top.
   2. If I’m pressing right, that means I am moving one to the right. That means the x value is 1 more than it was before.
      1. x = x+1;
   3. Now I need to draw a new tile at that location.
      1. drawTile(x, y, “yellow”); //or whatever color you chose
7. Now have them:
   1. Draw a maze
      1. Allow students to plan out their board on graph paper, if they like.
   2. Write if/else statements for pressing right, left, up, down.
      1. Teacher hint: be prepared for a lot of questions about when to add/subtract from x and y
8. As students finish, have them play each other’s games
9. End with a discussion:
   1. What went well?
   2. What didn’t?
   3. What did you learn?

## [Lab 5] If/Else Lab: Personalization

**Lab (computer), 90-120 Minutes (depending on group’s comfort with material)**

Concepts:

Replacing files in the resource folder

Learning Objectives:

Students will be able to:

Demonstrate an understanding of replacing files in the resource folder. Also demonstrate understanding of concepts covered throughout the week.

How to Assess (optional):

Observe student work on their computers

Materials:

* Computers with BlueJ and CorruptedFunctionalAPI (for each student)
* MS Paint or similar program that will allow resizing of pictures
* Projector showing teacher’s screen

Process:

1. Review Discussion:
   1. What have we discussed so far?
      1. Ints
      2. Strings
      3. If/Else
      4. etc
2. Let’s learn how to personalize the tiles!
3. Have students put pictures they brought on their computer.
   1. They can also use Free Clip Art, perhaps from: <http://www.school-clip-art.com/> or another age appropriate site
   2. If MS Paint is available, they can draw pictures.
4. Demonstrate how to resize pictures to 72 x 72 pixels on MS Paint or other available software
5. Demonstrate where to find the resources folder.
6. They will need to save their files with the color names. (Note: the pictures will show up best if the picture contains some of the color they are using it to represent).
   1. red.png
   2. purple.png
   3. yellow.png
   4. blue.png
   5. green.png
   6. light\_blue.png
7. Recompile and let them run the game!
8. Challenge them to make their own game using their new tiles.
   1. Ideas you can share with them:
      1. Draw a picture
      2. Maze game (What happens if you make it move UP when you press the down arrow? The student is in control!)
      3. Make it draw TWO tiles when you press a button. Or THREE! Or FOUR!
      4. Make a treasure map that someone has to run through. Can write a narrative using System.out.println(). Ex. What each color represents (island, x marks the spot, etc) and which order to visit in.
      5. Any other game they can think of!
9. Give them time to make games and play other student’s games.

# For Loops

## For Loops: Introduction

**Activity (non-computer), 30 minutes**

Concepts:

For Loops

Learning Objectives:

Students will be able to:

Identify or Describe a scenario in which a for loop would be useful.

How to Assess (optional):

Ask questions: What is a for loop used for? (Ans: To call a method multiple times). What problem does it help solve? (Ans: Write less repetitive code)

Materials:

* None

Process:

1. Take students outside.
2. Define a course the students will need to walk. At UW Bothell, walking the circle created by the sidewalk between UW1 and UW2 would work really well.
3. Tell them that you (the teacher) are the programmer. And all of them are the program. You wrote code for the program. The method in it is:
   1. takeStep();
4. Ask them what they think they need to do each time that method is called.
   1. Ans: Take one step.
5. Tell them that you need to write takeStep() enough times that they get all the way around the course and back to the beginning. The challenge is who can take the least number of steps (because want to have to write this method the least number of times).
6. Their task is to walk the course, counting their steps, and then come back and report to you how many steps they took. That will be the number of times that you will have to write takeStep() in the code. Encourage them to get the smallest number. They can make multiple attempts if they want.
7. When they get back, have them gather around you to discuss how many steps they took. For example, maybe they took 100 steps (I will use this example for the rest of this lesson plan. You may need to adjust the numbers you give the students based on the length of the course you give them and the numbers they provide you).
8. Tell them that you don’t want to write that one line of code 100 times! That’s too many! So, you’re going to write a loop.
9. Tell them the code now looks like this:
   1. For a count starting a 0, until the count reaches 20, takeStep().
10. Have them go around the circle and report back to you how many times they had to run the loop of 20.
11. When they get back, have them gather around you to discuss the number of loops. It should be around 5.
12. Tell them that’s STILL too much code to write!
13. This time, the code looks like this:
    1. For a count starting at 0, until the count reaches 100, takeStep().
14. When they get back, have them gather around and discuss the number of loops. It should be around 1!
15. Return to the classroom for final discussion.
    1. Tell them they were learning a programming tool called a for loop.
    2. This allows programmers to run the same line of code many times in a row without having to write out each time they want it called.
    3. Ask them to brainstorm where this might be useful in writing a game? In games they have played, have they had a character or object do the same thing over and over? Give examples!

## [Lab 6] For Loops Lab: Drawing Columns

**Lab (computer), 90 -120 Minutes (depending on group’s comfort with material)**

Concepts:

For Loops

Learning Objectives:

Students will be able to:

Demonstrate an understanding of for loops by drawing a column of tiles.

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Review Discussion:
   1. How did we draw the tiles on the screen last week?
   2. How many times did you have to write drawTiles()?
   3. What are for loops?
      1. A piece of code we can write that will call the same code for us a number of times. Makes less work for the programmer!
2. Let’s write code to draw a column of tiles together!
   1. What will a column look like? It will be a series of tiles all drawn at the same x position of the screen.
   2. I’m going to go into my buildGame() method and write the for loop for this.
   3. Let’s say I want a column of 10 tiles. This means I want a for loop that calls the drawTile() method 10 times.
      1. I need to tell the for loop 3 things:
         1. What number to start counting at.
            1. I am going to make an int called currentNum and start it at 0;
            2. Type on the board:

for (int currentNum = 1;

* + - 1. How many times to run the loop?
         1. I want it to run while the currentNum is less than or equal to 10
         2. Type on board:

currentNum <= 10;

* + - 1. What should happen to the currentNum number every time the loop is run? Should it go up? Down?
         1. Type on board:

currentNum = currentNum + 1);

* + 1. You should now have this written on the board:
       1. for(int currentNum = 1; currentNum <= 10; currentNum + 1)
    2. Add the curly brackets with space to type in.
    3. Now, I want it to call drawTile() each time the loop runs.
       1. Type: drawTile(
       2. Ask class: What parameters do I need to give it?
          1. X location:

Since I want it to always draw when x = 8, I will put an 8 there.

* + - 1. Type: drawTile(8,
         1. Y location

I want it to change each time. This is why for loops are magic! The currentNum is going to start at 1 and go up to 10. I know it changes each time. I want the y location to be whatever the currentNum is each time.

* + - 1. Type: drawTile(8, currentNum,
         1. Color
         2. Have class pick a color. I’ll use blue here.
      2. Type: drawTile(5, currentNum, “blue”);
    1. Talk them through what this does. SLOWLY. Example for one loop talk through below. You should plan on doing this at least 3 times.
       1. To start, the currentNum is 1. The program is going to check: if that less than or equal to 10? Yes! Great, run the code.
       2. It goes into the code and draws a tile at 8, 1. How did it know to draw at 1? <have class answer>
       3. It reaches the end of the loop and adds 1 to the currentNum. Why? <have class answer>
       4. Now it goes back to the top. What is currentNum? It’s 1. It checks if that is less than or equal to 10. Is it? Yes! Great, run the code
       5. Etc.
       6. End with: When does it stop? What will the currentNum be?

1. Let’s run this code and try it!
   1. Demonstrate it works.
2. Give the students the task to draw a board. Provide time to experiment.
3. Discuss:
   1. Was this easier or harder?
   2. How much code did you have to write compared to before?
   3. What are the limitations on this? Hint: Color!

## [Lab 7] For Loops Lab: Make a Board!

**Lab (computer), 90-120 Minutes (depending on group’s comfort with material)**

Concepts:

For Loops

Learning Objectives:

Students will be able to:

Demonstrate an understanding of for loops by drawing a board of tiles.

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Review Discussion:
   1. What do we use for loops for?
   2. What was the limitation when we drew a column using a for loop? (color)
2. Let’s practice writing for loops again. This time, try using this method:
   1. getRandomColor()
   2. What do you think this method does? How can you use it?
3. Give students time to create a board using a series of for loops.
4. Discuss:
   1. How was this better than before?
   2. Are for loops getting easier?
   3. How many for loops did you have to write?
5. Let’s try creating our board using only one for loop!
   1. How can we do this? By drawing columns instead of tiles!
6. Challenge them to try figuring this out on their own. Experiment with a for loop using the drawColumn() method.
7. After most of the class has finished, demonstrate how to do this using the projector (reinforces the concept and helps those who are still struggling to catch up).
8. Discuss:
   1. How many times did you run the loop to create the whole board?
   2. What number did you start currentNum at?
   3. Were for loops useful? Which do you like more? Writing all the drawTile() methods? Or using a for loop?
      1. If students like writing drawTile() more, ask them if they would change their opinion if they had to write it 100 times? 500 times? 1000 times?

# Gameplay

## [Lab 8] New Methods Lab: Laser and Matching Tiles

**Lab (computer), 120 Minutes (depending on group’s comfort with material)**

Concepts:

Writing code to check for matched tiles and delete them.

Learning Objectives:

Students will be able to:

Demonstrate an understanding of if/else and gameplay mechanics.

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Review Discussion:
   1. What concepts were covered last week? Briefly review/discuss.
2. Let’s talk about our tile game! Think back to the Corrupted game we played the first day (Note: Maybe let students play it again for 5 – 10 minutes).
   1. What is gameplay like?
   2. What happens when you use the laser?
   3. What buttons do you press?
3. Let’s write some code to do this!
   1. Have students start by writing code for drawing the laser and moving it.
      1. Tell them they have these 3 new methods
         1. drawLaser()
         2. moveLaserUp()
         3. moveLaserDown()
      2. They still have access to the old methods:
         1. pressingUp()
         2. pressingDown()
         3. Etc
   2. Tell them to write their buildGame() and updateGame() methods. Try to get the laser moving. Give them 10-20 minutes to work on this on their own. Then, bring the class’ attention to the front of the room and on the projector, demonstrate for them how you would write it. This will allow students to verify their work and fix any mistakes they may have made.
4. Now talk to them about what happens when they press the right arrow during gameplay. Have them help you make a list on the board. It should include:
   1. Draws a new tile on the same line
   2. Connect and/or delete the matching tiles
   3. Change the laser color
5. Tell them to do this, you need to be able to get the number of matching tiles. We’re going to create an int variable to track this!
   1. Tell them you are going to create an int called “matches”
   2. You are going to make matches equal the number of matching tiles by using the method getNumberofMatchingTiles()
   3. Write this code on the screen for them to see.
   4. Tell them they would want to compare this number to the number of tiles they want to be matched (their choice of how many!). To do this, they can compare with <, >, ==. Don’t demonstrate how to write this code, but write the comparison operators on the board for them to reference.
6. Now inform them they can use the following methods to write what happens when they press right:
   1. drawNewTileFromLaser();
   2. getNumberOfMatchingTiles();
      1. you showed them this one
   3. deleteMatchingTiles();
   4. setNewLaserColor();
   5. setTilesInMatchSet(int num);
      1. How many tiles to match before the set deletes?
7. Give them time to work on this on their own.
8. If some of the students finish early, challenge them to play with different numbers of matches.
9. When most of the class has finished, bring them together and demonstrate writing the code on the projector.
10. Final discussion:
    1. What was most challenging about this?
    2. What numbers of matches did you experiment with? What was fun? What was not fun?

## [Lab 9] Gameplay Lab: Win Conditions

**Lab (computer), 90-120 Minutes (depending on group’s comfort with material)**

Concepts:

Gameplay mechanics.

Writing a new method and calling it from updateGame()

Learning Objectives:

Students will be able to:

Demonstrate an understanding of method calls.

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Discussion:
   1. Think of some of your favorite games.
      1. How do you win?
      2. How do you know you’ve won?
   2. Think of our tile games.
      1. How do you win?
         1. Hint: We haven’t made a way to win yet!
      2. What are some ways we could make it so you could win?
      3. What should be the win conditions?
         1. Note: Try to steer the conversation towards counting matched tiles.
2. First things first. we need a way to count the tiles!
   1. Tell them they can use the following methods:
      1. drawCounter()
      2. setCounterValue(int count) //the # of tiles to match to win
      3. updateCounter()
   2. Give them time to implement and experiment with the counter number.
3. Let’s write some code. Tell them they will be writing a new method today!
   1. Instruct them to set it up the same way they would buildGame() with “public” and “void.”
   2. The difference is that this method will be called “checkGameWon()”
      1. Tell them there is a variable that the game keeps track of called “tilesNeededToWin”. It is an int, and it starts at the number they set the counter to. Each time tiles are matched, it goes down by the number of matched tiles. Tell them they can use this variable to figure out if the game is won. Let them experiment to see if they can figure out how! If the game is won, they should call this method:
         1. winGame()
   3. To try the method out, they will need to call it. They should call it from within updateGame(). Depending on how confident they are, you may want to demonstrate calling this method for them on the projector.
4. Give them time to experiment. Change the number of tiles needed to win. Play each other’s games!
5. Discussion:
   1. What number of tiles was fun? What was not fun?
   2. If you were making this game for younger kids, would you change the number? What about for college students? Is it important to consider who will be playing the game?

## [Lab 10] Gameplay Lab: Cleanup the game!

**Lab (computer), 90 Minutes**

Concepts:

Adding methods and calling them from updateGame()

Learning Objectives:

Students will be able to:

Demonstrate an understanding of method calls

How to Assess (optional):

Observe student work on their computers

Materials:

* A projector showing the teacher’s screen.
* A computer with BlueJ for each student.
* The CorruptedFunctionalAPI on each computer

Process:

1. Review Discussion:
   1. Yesterday we learned about writing a new method.
      1. How did we do it?
      2. What was the same? What was new?
2. Let’s add a new gameplay element to the game! The tiles shifting left.
   1. Discuss: What would this do to the game? More fun? Harder?
   2. The methods they can use are:
      1. startTimerForTiles(int seconds) //goes in buildGame() to create timer
      2. isTimeToShift() //returns true when it’s time to shift
      3. shiftTilesLeft() //shifts all the tiles left
   3. Inform the class, they can compare isTimeToShift() to true or false. They can use the comparison operators of == and !=
   4. Give them time to experiment.
3. When most of the class seems to have finished, demonstrate writing it on the projector (reinforces concept).
   1. Emphasize this concept: Should it be in an if statement? Or else if?
      1. Let them brainstorm and guess.
      2. Discuss: It should be if statement because we want it to check the timer every time update is called, not just when the user isn’t pressing up, down, or right.
4. Discuss: Notice anything about the gameplay? What happens when the tiles get to the laser? Does it cover them? Does it look good? Nope! Let’s fix it!
   1. Use the method:
      1. deleteMissedTiles()
5. Give them time to play with these new methods and create a fun game.
6. Discuss:
   1. What settings did you come up with that were fun?
   2. Who did you make this game for (what audience)?

## Final Lab: Make a Game or 2!

**Lab (computer), 120 Minutes (depending on group’s comfort with material)**

Concepts:

Variables, parameters, method calls, if/else, for loops

Learning Objectives:

Students will be able to:

Demonstrate an understanding of computer science concepts by implementing at least one game using the API

How to Assess (optional):

Observe student work on their computers

Materials:

* Computers with BlueJ and CorruptedFunctionalAPI (for each student)
* Projector showing teacher’s screen

Process:

1. Today’s the day we’ve all been waiting for! You get to use everything you’ve learned to write your own game! Or 2! Or 3!
2. Ask if there are any questions or topics they’d like to review.
3. Provide them with a list of methods they learned that they can use. Possibly the java doc? Or just write them on the board.
4. Tell them their games must have the following:
   1. Tiles
   2. Win Condition
   3. Awesome name
   4. Instructions (use System.out.println)
5. When they finish their game, have other students play it. Make changes if they want. But most importantly, HAVE FUN!
6. End of project discussion:
   1. What were some of your favorite games that you saw today?
   2. What did you like about them?
   3. What was your favorite thing that you learned? Did you use it in your game?

## Sample Camp Schedule: Week 1

**Monday**

Introductions & Class Rules (15)\*

Ice Breakers (30)

Computer Science: What is it? (15)

Time to play Corrupted and Discuss Game Elements (30)\*

Variable Types: Int and String (15)

Writing Strings to Console: Intro to Programming (30)

Method: What Does a Method Look Like? (60)

Total: 3 hrs, 15 mins

**Tuesday**

Intelligent Piece of Paper (30)

Method Mad Libs w and w/out parameters(30 - 45)

Drawing Tiles with Parameters (120)

Total: 3 hrs, 15 mins

**Wednesday**

BuildGame and UpdateGame (30)

If/Else Intro Game (45)

If/Else Lab (60)

BuildGame and UpdateGame Extensions (with if/else and with bugs) (20)

Instance of an Int (30)

Total: 3 hrs, 5 mins

**Thursday**

If/Else Lab Extension: Using Variable Declarations (60)

BuildGame and UpdateGame Extensions (go outside in teams) (30)

If/Else Lab: Maze Game (120)

Total: 3 hrs, 30 mins

**Friday**

Game Mechanics: Win State (60)

If/Else Lab: Personalization (120)

Preview for what we will learn next week (15)\*

Total: 3 hrs, 15 mins

## Sample Camp Schedule: Week 2

**Monday**

Variables to Track Info: Variable Tag (60)

New Methods Lab: Laser and Matching Tiles (120)

Total: 3 hours

**Tuesday**

Variables: Box Variables (45-60)

For Loops: Introduction (30)

For Loops Lab: Drawing Columns (120)

Total: 3 hrs, 15 mins

**Wednesday**

For Loops Lab: Making a Board (90)

If Statements and Flow Control: The Imp Computer (90)

Total: 3 hrs

**Thursday**

Algorithms: The Swap Puzzle (60)

Gameplay Lab: Win Conditions (120)

Total: 3 hrs

**Friday**

Gameplay Lab: Cleanup the Game (90)

Final Lab: Make a Game! Or 2! (120)

Total: 3 hrs, 30 min

\*No specific lesson plan written