CS 2103: Class 1

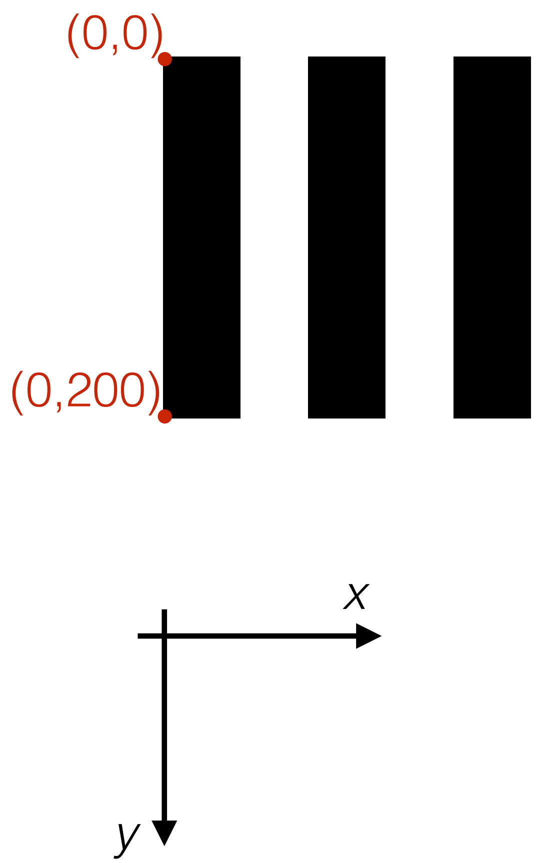
Jacob Whitehill

Staff introductions

Exercise: Band

Exercise: Band

- Consider the bands shown below:



Three identical black bands with height of 200, width of 20.
Each gap between adjacent bands has a width of 20.

Exercise: Band

- Consider the bands shown below:

- Assume you are given the following class & method:

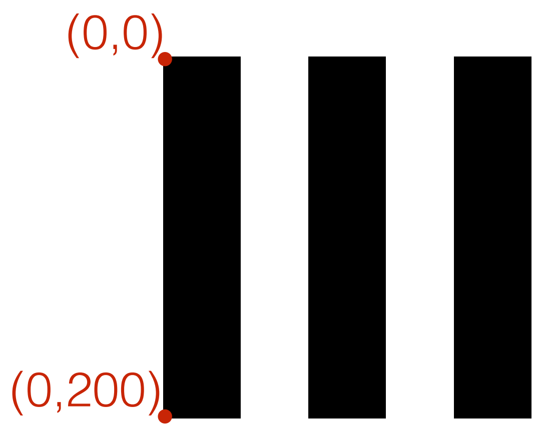
  - ```
    // Simple coordinate-pair class
    class CoordPair {
      int _x, _y;
      public CoordPair (int x, int y) {
        _x = x;
        _y = y;
      }
    }
    ```

  - ```
    // Draws & fills a black rectangle whose corners are given,
    // in sequence, by the specified array of coordinate-pairs.
    void drawRect (CoordPair[] coordinates) { ... }
    ```

- Write a Java method that draws the bands above:

```
void drawBands () {
  // IMPLEMENT ME ...
}
```

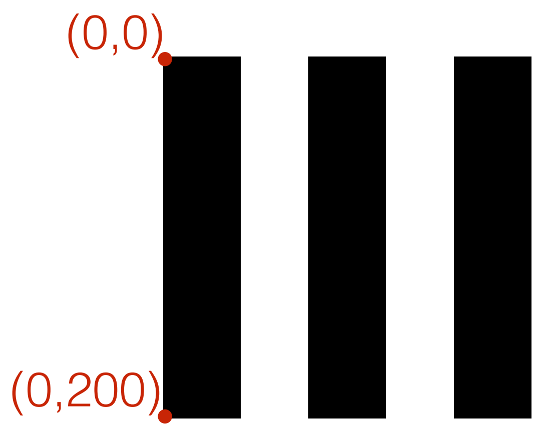
Solution(s): Band



Three identical black bands with height of 200, width of 20.
Each gap between adjacent bands has a width of 20.

```
void drawBands () {
  for (int i = 0; i < 3; i++) {
    final CoordPair[] coordinates = new CoordPair[4];
    coordinates[0] = new CoordPair(i*40, 0);
    coordinates[1] = new CoordPair(i*40, 200);
    coordinates[2] = new CoordPair(i*40 + 20, 200);
    coordinates[3] = new CoordPair(i*40 + 20, 0);
    drawRect(coordinates);
  }
}
```

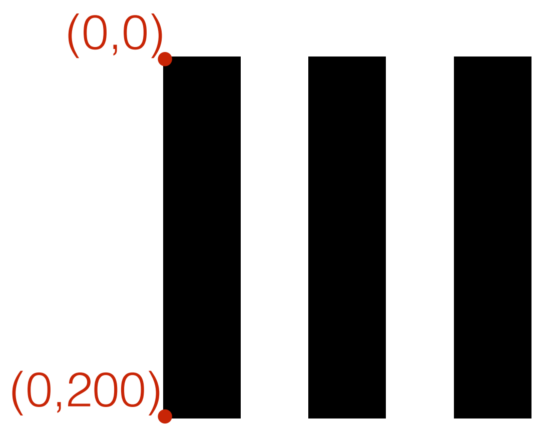
Solution(s): Band



Three identical black bands with height of 200, width of 20.
Each gap between adjacent bands has a width of 20.

```
void drawBands () {
  drawRect(new CoordPair[] { new CoordPair(0, 0), new CoordPair(0, 200),
                             new CoordPair(20, 200), new CoordPair(20, 0) });
  drawRect(new CoordPair[] { new CoordPair(40, 0), new CoordPair(40, 200),
                             new CoordPair(60, 200), new CoordPair(60, 0) });
  drawRect(new CoordPair[] { new CoordPair(80, 0), new CoordPair(80, 200),
                             new CoordPair(100, 200), new CoordPair(100, 0) });
}
```

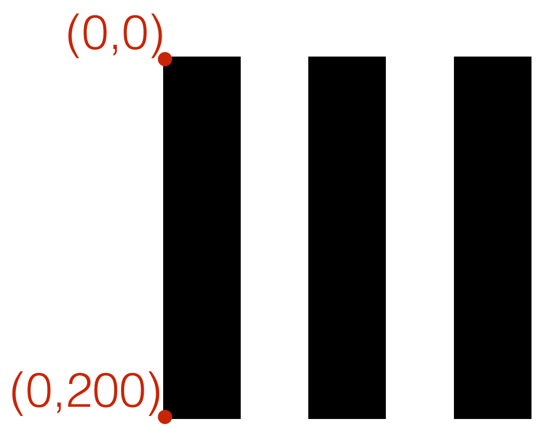
Solution(s): Band



Three identical black bands with height of 200, width of 20.
Each gap between adjacent bands has a width of 20.

```
void drawBands () {
  int startX = 0;
  for (int i = 0; i < 3; i++) {
    final CoordPair[] coordinates = new CoordPair[4];
    coordinates[0] = new CoordPair(startX, 0);
    coordinates[1] = new CoordPair(startX, 200);
    coordinates[2] = new CoordPair(startX + 20, 200);
    coordinates[3] = new CoordPair(startX + 20, 0);
    drawRect(coordinates);
    startX += 40;
  }
}
```

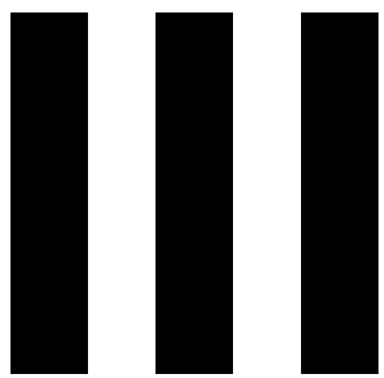
Solution(s): Band



Three identical black bands with height of 200, width of 20.
Each gap between adjacent bands has a width of 20.

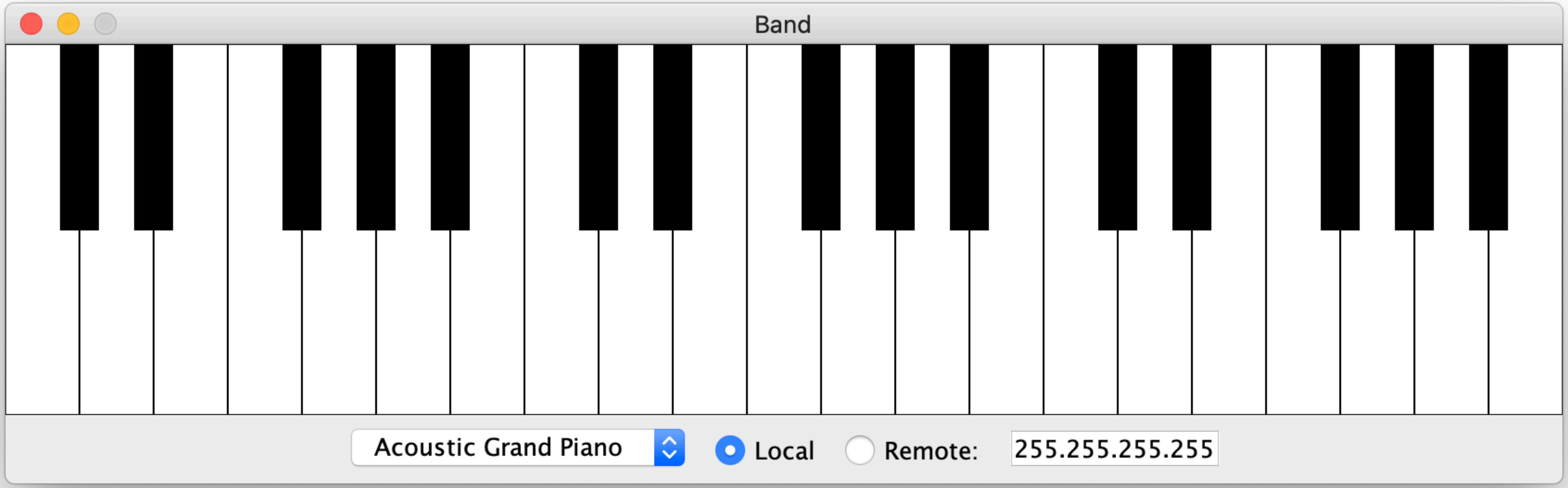
```
void drawBands () {
  final int BAND_WIDTH = 20;
  final int BAND_HEIGHT = 200;
  final int GAP = 20;
  final int X_INCREMENT = BAND_WIDTH + GAP;
  final int NUM_BANDS = 3;
  int startX = 0;
  for (int i = 0; i < NUM_BANDS; i++) {
    final CoordPair[] coordinates = new CoordPair[4];
    coordinates[0] = new CoordPair(startX, 0);
    coordinates[1] = new CoordPair(startX, BAND_HEIGHT);
    coordinates[2] = new CoordPair(startX + BAND_WIDTH, BAND_HEIGHT);
    coordinates[3] = new CoordPair(startX + BAND_WIDTH, 0);
    drawRect(coordinates);
    startX += X_INCREMENT;
  }
}
```

Exercise: Band



Exercise: Band

Project 1: Band



Introduction to CS 2103

InstructAssist

- CS2103 2020 B-Term will use Canvas as the learning management system.
- You are required to read the entire course syllabus and course policies.
- You are also required to join the course Slack channel.
- **Make sure** you check the Canvas page and Slack channel at least 1/day.

Learning goals of CS2103

1. Learn how to use **object-oriented programming (OOP) principles** to design and implement complex software systems to be:
    - correct
    - intuitive
    - extensible
    - safe
    - efficient

Learning goals of CS2103

2. Learn techniques for **OOP in Java**, including:
    - objects
    - message passing/method calls between objects
    - interfaces (as types, contracts, callbacks)
    - classes (concrete, abstract, inner, anonymous)
    - access restriction (private, protected, public, package-private)
    - generic types for flexible type-safety

Learning goals of CS2103

3. Learn the interface and implementation of the **canonical data structures** used in computer programming:
    - Lists (array-based and linked)
    - Stacks & queues
    - Trees
    - Hash tables
    - Graphs
    - Heaps

Learning goals of CS2103

4. Solve interesting & challenging programming problems:
    - data modeling — how to represent and connect the important data
    - algorithmic — how to achieve the computational goal using the available resources

Learning goals of CS2103

5. Gain exposure to more advanced computer programming techniques such as:

    - Graphical user interfaces & event-driven programming
    - Quantitative analysis of canonical data structures and fundamental algorithms
    - Graphs & social networks
    - Context-free grammars

CS2103 Road Map

Time

intro to OOP
intro to event-driven programming
callbacks
objects
classes
interfaces
casting (up, down)
intro to abstract data types (ADTs)
Java generic types
ArrayList
intro to asymptotic algorithmic analysis
LinkedList
inner classes
hashtables and HashMap
hashCode() and equals()
graph
stack
queue
Java generics type bounds
graph search (BFS, DFS)
trees
binary search trees
heaps
expression trees
anonymous classes
lambda expressions
design patterns

These are some **key programing constructs** for object-oriented programming.

CS2103 Road Map

Time

intro to OOP
intro to event-driven programming
callbacks
objects
classes
interfaces
casting (up, down)
intro to abstract data types (ADTs)
Java generic types
ArrayList
intro to asymptotic algorithmic analysis
LinkedList
inner classes
hashtables and HashMap
hashCode() and equals()
graph
stack
queue
Java generics type bounds
graph search (BFS, DFS)
tree
binary search tree
heap
expression tree
anonymous classes
lambda expressions
design patterns

These are some of the **canonical data structures** used in computer programming.

CS2103 Road Map

- OOP **facilitates** the implementation of canonical data structures for computer programming.

CS2103 Road Map

- OOP **facilitates** the implementation of canonical data structures for computer programming.
- Higher-level OOP (for larger projects) also **uses** these data structures to model complex relationships among the data.

CS2103 Road Map

Time

intro to OOP
objects
classes
interfaces
casting (up, down)
intro to abstract data types (ADTs)
Java generic types
ArrayList
intro to asymptotic algorithmic analysis
LinkedList
inner classes
hashtables and HashMap
hashCode() and equals()
graph
stack
queue
Java generics type bounds
graph search (BFS, DFS)
tree
binary search tree
heap
expression tree
intro to event-driven programming
callbacks
anonymous classes
lambda expressions
design pattern

Hence, our learning of OOP design techniques and the implementation of these data structures will be **interlaced**.

Grading

- Course grade:
  - 60% programming projects (1/week)
  - 40% exams (1 midterm, 1 final)

Grading

- Exams:
  - **Midterm**: Monday, November 22
  - **Final**: Thursday, December 16
- Length: 1 hour 30 minutes (4:00-5:30pm)

Grading of programming projects

- CS 2103 uses a **combination** of human grading and automatic grading (via scripts).

- Automatic grading:

Grading of programming projects

- CS 2103 uses a **combination** of human grading and automatic grading (via scripts).

- Automatic grading:

  - Fast ==> earlier feedback to you.

Grading of programming projects

- CS 2103 uses a **combination** of human grading and automatic grading (via scripts).
- Automatic grading:
    - Fast ==> earlier feedback to you.
    - Reliable: automated test cases are more reliable than manually reading through code.

Grading of programming projects

- CS 2103 uses a **combination** of human grading and automatic grading (via scripts).
- Automatic grading:
  - Fast ==> earlier feedback to you.
  - Reliable: automated test cases are more reliable than manually reading through code.
  - Consistent across students.

Grading of programming projects

- CS 2103 uses a **combination** of human grading and automatic grading (via scripts).
- Automatic grading:
    - Harsh: one extraneous semicolon can result in a big grade penalty.

Grading of programming projects

- All submitted projects will be graded first by an auto-grading script.
- Then, an SA/TA/me will manually check where many points were lost.
- We usually give some credit for a “reasonable attempt”.
- Whenever possible, we design the test cases to be granular — not just all-or-nothing.
- We also manually grade submitted projects for OO design.

Grading of programming projects

- Help us help you!
  - If a programming project description says “X is important” then **make sure you do X**.
  - **Always run your code against the test code we provide**.
    - If your code doesn’t compile against our tester, then your score will be very low.

Collaboration

- You are **highly encouraged** to collaborate as part of a team (**preferably 2**, at most 3 people) on all projects.

- Every member of the team is responsible for knowing about **all** aspects of **all** projects.

Collaboration

- Collaboration is encouraged:
  - On programming projects
  - During in-class exercises
- Collaboration is forbidden:
  - On exams

Collaboration

- On programming projects **within the same team**:
  - Talk freely about the concepts, challenges, and strategies for solving those challenges.
  - Share code freely — because you are working on it together.

Collaboration

- On programming projects **between teams**:
  - Talk freely about the concepts, challenges, and strategies for solving those challenges.
  - You may **not** share code in any way (email, visually, verbally, etc.).

Help

- You are welcome and encouraged to ask questions at any time.
  - In-class: raise your hand, or type on Slack.
  - Outside of class: via Slack channels & office hours.

Help

- You are welcome and encouraged to visit office hours (me, the TA, and/or the SAs).
- Office hours will be either in-person or on Zoom.
- See the course syllabus for a schedule of who offers office hours when.
  - We may occasionally post updates to Slack #general.
- Office hours start this **Tuesday**, October 26.

Respect

- Everybody makes mistakes — that includes me, you, and the TA/SAs.
- I require every member of the course (students+staff) to show respect to everyone at all times.
- Problems with other students: see me.
- Problems with TA/SAs: see me.
- Problems with me: see Prof. Craig Wills, head of dept.

Introduction to event-driven programming

Control flow

- In contrast to many previous Java programs you've probably written, Project 1 is an **event-driven program**.

- This means that the control flow is not fixed; it is driven by user input events (mouse, keyboard, etc.).

- **Control flow**: the order in which statements of code are executed and methods are called.

Control flow

- Consider the following (silly) code:

```
public static void main (String[] args) {
  f();
  for (int i = 0; i < 10; i++) {
    g(i);
  }
}
void f () {
  System.out.println("Yo, it's me, f!");
}
void g (int num) {
  System.out.println("`sup, it's g: " + num);
  h();
}
void h () {
  // nothing
}
```

Control flow

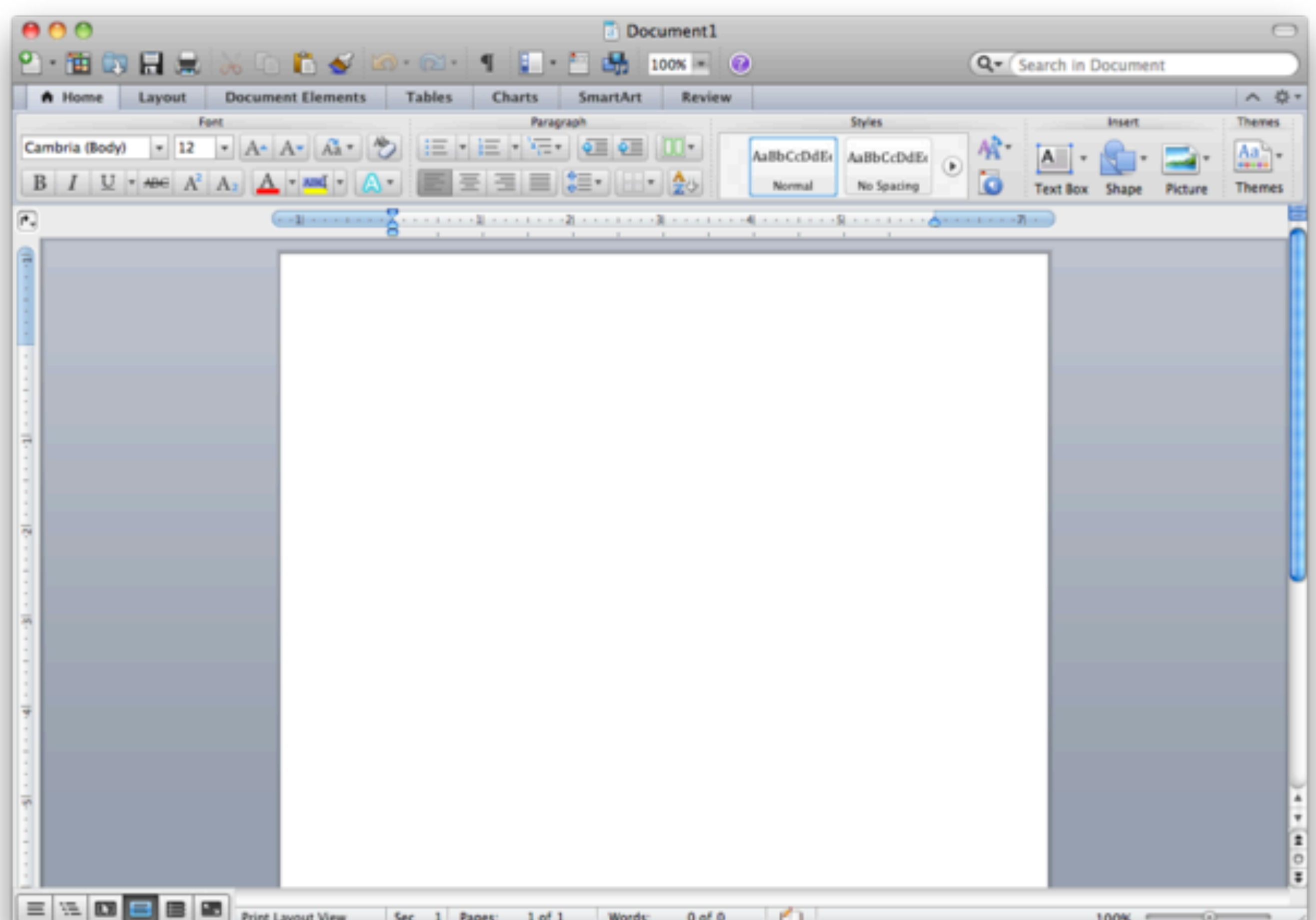
- Consider the following (silly) code:

```
public static void main (String[] args) {
  f();
  for (int i = 0; i < 10; i++) {
    g(i);
  }
}
void f () {
  System.out.println("Yo, it's me, f!");
}
void g (int num) {
  System.out.println("`sup, it's g: " + num);
  h();
}
void h () {
  // nothing
}
```

- The control flow can be completely inferred from the code itself — it's clear that `f` is called only once; `h` is called once per call to `g`; `g` is called 10x; etc.

Event-driven programming

- In Microsoft Word, the program isn't "doing" anything for most of the time.
  - It's just **waiting** for the **user** to do something (e.g., type something, click the mouse).



Event-driven programming

- In interactive applications, the control flow can be **variable**.
  - When will the user click a button?
  - When will the user move the mouse?
  - When will the user press a button on the keyboard?
- It may also be desirable to update the GUI at a regular rate (e.g., ~60Hz).
  - Update position of different objects on the game board.

Event-driven programming

- Event-driven programs are often structured very differently from purely “computational” programs.

- Instead of method `main` calling `g` calling method `h`, etc., an **event-driven** program will tell an event framework (Swing, in our case) what method to call when a specific event occurs, e.g.:

```
public static void main (String[] args) {
  Swing.callMethodIfMouseIsClicked(mouseWasClicked); // bogus, but the right idea
}
public void mouseWasClicked (int x, int y) {
  System.out.println("Ooh! The mouse was clicked at: " + x + " " + y);
}
```

- The mouseWasClicked method is sometimes called an **event handler**, **listener**, or a **callback**.

Event-driven programming

- In event-driven programming, the control flow can vary significantly from one run of the program to another.

- Method calls are made less frequently by the application developer and more frequently in response to external events (e.g., mouse click, button press).