Mini-Assignment Due: Monday, September 30th, 11:59pm
Early Due Date: Friday, October 4th, 11:59 pm*
On-Time Due Date: Sunday, October 6th, 11:59 pm*
Late Due Date: Tuesday, October 8th, 11:59 pm*

* In addition, please note that the SunLab closes at 10:00 pm on Friday and Sunday

To run demo: cs0150_runDemo FruitNinja
Demos may not work well over ssh! Try FastX or the Sunlab.
To install: cs0150_install FruitNinja
To handin: cs0150_handin FruitNinja

Silly premise

Jim is back with his next elaborate prank! After getting his black belt in karate, Dwight decides to try learning how to use a katana. Michael is tossing up fruits for Dwight to slice out of mid-air for practice. Jim thinks it would be funny to make things more difficult and decides to throw a few bombs into the mix. If the bomb is cut, the entire office will be demolished, and no one will have paper in all of Scranton! Help Dwight master his katana skills and keep the office from destruction!
**Collaboration Policy Reminder**

From the [collaboration policy](#):

Collaboration on projects is not allowed, except for mini assignments and in discussion sections with your section TAs. Otherwise, **no collaboration is allowed on any code whatsoever. You should never, ever look at another student’s code.** You may talk about high-level design concepts discussed in the mini-assignments, but you may not discuss the implementation and debugging of code for projects with anyone except the course staff.

**Important:** **you may absolutely discuss general** (i.e., not assignment-specific) **CS15 concepts with anyone,** including other current students. The following falls into this category:

- Going over CS15 lecture slides, our (non-assignment) handouts, Javadocs, etc.
- Discussing object-oriented programming concepts, such as polymorphism
- General syntax questions. For example, “How do I declare an instance variable?”
- How to work remotely, and how to move and hand in files

Note that in each case, any examples used must be from the lectures or your own creativity – you may not discuss how even broad design concepts like inheritance pertain to a specific assignment.

**New Concepts Covered**

- Polymorphism
- Inheritance
- Interfaces
- Switch Statements

**Mini-Assignment**

For FruitNinja and all future programming assignments, we will also have a corresponding mini assignment. The mini-assignment questions will get you started on carefully planning the design of your program, and will greatly help with the coding process. You will review and discuss your design during section. Make sure to read this handout in full before starting the FruitNinja mini-assignment.
Assignment Specifications

Before you read this section, make sure you play around with the demo! Better yet, have it open while you read this and play along. Doing so will make understanding the assignment much easier.

Your assignment is to write a program that displays the CS15 version of a game of FruitNinja with a blade, bombs and four types of fruit.

- You should have a blade and it should have a name to go along with your fun and cool project. You DO NOT need to write your own blade class -- this is done for you. Check out the support code docs to see what methods you can call with your blade.
- Apples, lemons, pears, peaches and bombs should randomly be launched up as the game progresses. These fruits should be washed and ripened when they are launched.
- You should be able to chop all of these with your blade.
  - If a fruit is chopped, the score should increase according to the fruit that you chopped. (Each type of fruit should have a different score). There should also be a splash after it has been chopped.
  - If a bomb is hit, it should explode and the game should end.
- The game should keep track of the score.
  - To do this, you only need each fruit or bomb to keep track of how many points it should be worth when chopped, and return that score value in the appropriate method. We will handle the actual score incrementing for you!

Do First

- Read this handout carefully. Do it, and then do it again.
- Read through the support code doc and use it while you are coding.
- Make sure you understand the design discussion mini-assignment. Review lectures slides and/or go to hours if you have questions about the mini-assignment or project design.
  - Specifically, make sure you understand the containment and inheritance/interface relationships.
  - What do different types of fruit have in common with each other? What do fruits have in common with bombs? Think about how you can factor out these similarities using polymorphism/interfaces/inheritance.
- Look at the demo and play around with it.
Fruits for Thought on Polymorphism

Apple, Lemon, Peach, and Pear

What do your Fruits have in common with each other? What do your Fruits not have in common with each other? How can you use inheritance to factor out code into superclasses?

Fruits and Bombs

Think about what common actions happen to fruit and bombs and what they don't. (Hint: Maybe it's time to write a short interface? Fruits and Bombs are both Choppable but different things should happen when each of these are chopped...)

Programming Tips & Working with Support Code

Check out which support classes are abstract to get a feel for the kinds of classes you’ll need to write on your own. Remember that you cannot directly instantiate an abstract class but you can instantiate a concrete subclass of an abstract class!

Support Code Documentation

Just like previous projects, we’ve written the support code documentation in Javadocs format, which is linked again here. Each class has their own page, and their methods are outlined on that page as well. You’ll be using the real Javadocs from the next project onwards, so get ready for that!

Setting Up the GUI (Graphical User Interface)

This time, the only stencil class you’ll be receiving with the install script is the App class. In this class, you'll only have one method: the start(Stage stage) method in the App class. This method starts the application, and is similar to a constructor for the App class, but is a special method used to start a graphical application. You can instantiate your top-level class here, and you do not need to use the Stage parameter.

Check out the class FruitNinjaFrame in the support code to see how to add the game frame to the scene and set up the GUI.

The “Game” Class

Your game class will contain most of your game logic. This is where you will deal with starting the game, launching fruit, chopping fruit, fallen fruit, incrementing score, etc...

Switch Statements
Be sure to review the lecture slides on switch statements in the Math and Making Decisions lecture! You will need to use a switch statement to randomly launch fruit and bombs onto the screen during your game of FruitNinja. Remember, the objects appearing on the screen need to be random and you might find the Math.random() method in the Javadocs helpful for generating a random number!

Reacting to Mouse Interaction

In this program you will need some way to detect when the blade, dragged by the mouse, has sliced a fruit or bomb. You don’t need to handle the mouse/collision interaction yourself—the support code does that for you. You only need to decide what happens when the mouse comes in contact with a game object.

If the user drags the blade through a fruit, it should be sliced. If it’s a bomb, the bomb should explode and the game should end. A method will be called each time the blade comes in contact with an item on the screen (see the support code docs for details).

Minimum Functionality Requirements

MF Policy Summary: In order to pass CS15, you will have to meet minimum functionality requirements for all projects. If you don’t meet them the first time around, you may hand the project in again until you succeed, but you will keep your original grade. MF requirements are not the same as the requirements for full credit on the project. You should attempt the full requirements on every project to keep pace with the course material. An ‘A’ project would meet all of the requirements enumerated in the assignment specification section of the handout and have good design and code style.

To meet minimum functionality for FruitNinja:

- The game must have a blade that the user can move
- Bombs and at least one type of fruit must launch into the game
- Both bombs and at least one type of fruit must be react by being “chopped” when the blade contacts them

Coding Incrementally: Follow This Plan!!

Just as you designed this program in steps, you should code it in small parts. Write some code, get it to compile and run correctly, and then move on.

Here are some suggested steps for coding incrementally:

1. Start by writing a top-level class, and make the game show up in the frame. We highly recommend getting something to show up on the screen before doing anything else.
Java might ask you to define certain methods before letting you compile to achieve this step. We recommend returning 0 for `onBladeContact()` and `null` for `launchItem()` just for now to display the GUI before implementing said methods.

2. Set up the fruit and get them to show up correctly in `launchItem()`, one type at a time. Make sure the fruits are washed and ripened when they are launched.

3. Set up the bombs and get them to show up with the fruit.

4. Implement blade slicing in `onBladeContact()`. What should happen when a fruit is chopped vs. when a bomb is chopped?

5. Complete remaining game functionality.

Make sure you are not writing repetitive code! Your objects should be generic enough to make adding a different fruit very easy. You might have to alter parts of the program’s design and make modifications to code you’ve already written as you work. However, the more time you spend on the design phase before you begin coding, the fewer changes you will have to make later.

### Running Your Code

To run the program you must first `cd` (change directory) into `~/course/cs0150/FruitNinja` directory, then compile as you normally would by running `javac *.java` in your terminal. Run your program by typing `java FruitNinja.App`

### Handing In Your Code

In order to hand in your code, run the handin script at the top of this handout. The script will list all of the files you are about to hand in and will prompt you to confirm. Once you’ve confirmed, you will receive an email stating that the handin was successful.

Note: The email is your receipt or proof that you’ve handed in the assignment successfully, so please do not delete it! *If you did not receive an email, we did not receive your handin.* You can run this script as many times as you would like; however, once you run the script, all past handins for this assignment are overridden.

This means that if you run the script after the on-time deadline at the top of this handout, the project will be marked *late*, even if you handed in an earlier version on time. We will *not* accept emailed submissions or handins after the late deadline. If you are not finished by the late deadline, it is always better to hand something in for partial credit than turn in nothing at all.