----->1<-----

Create Launcher, Display class

----->1.1<-----

Create Display Class

/\*Creates a window where we can put our canvas \*/

- Initialize a JFrame with size, visibility etc.

- Declare a constructor with params {width, height, title}

----->1.2<-----

Create Launcher Class

/\*Has the main() method of the application\*/

///Demo --> Call a new Display([]params) object in the Launcher

///A new Window should appear

----->2<-----

Creating Canvas, Showing images on Canvas

----->2.1<-----

Create Canvas object in Display class

- Initialize a Canvas object

- Set it's prefered, minimum ,maximum size with dimension parameter

- Add the canvas into the frame

- Pack the frame to ensure visibility

///Demo --> Run the application

///Nothing changes because nothing is drawn on the canvas

----->2.2<-----

Create a Game class

/\*It will hold the main class\*/

- Create a constructor with params {title, with, height}

- Initialize a new Display object

///Demo --> Call a new Game([]params) object in the Launcher

///Assure everything is display correctly

----->3<------

Creating Threads and Loops

----->3.1<-----

Short and understandable definition of a thread:

"When you write a code, the comiler creates a program.

Each thread is a separate mini program that runs while our

main thread runs so we use our CPU to the maximum capacity

in order to process two things at once."

----->3.2<-----

Implement Runnable interface

/\*It allows the class to be put on a separate thread\*/

- Define a run() method that comes from the interface

----->3.3<-----

Define a synchronized start method to start the thread

- Define the start() method

- When initilizing the new thread we should specify the

class we are working on

Define a synchronized stop method to stop the thread

- Define the stop() method

- Call the join() method on the thread in order to stop it

from working

----->3.4<------

Work with the run() method

- Define an init method above the run method and call it

in the run method

- In the run() method:

- Define a game loop with while-loop

- Define tick() method

- Define render() method

- Call stop() method after the loop

- Move the display initialization to the init() method

- Create a new title field and initialize it in the constructor

///Demo-> Run the game and nothing should happen

///Explanation: The Display object is in the init() method

///that is called in the run() method that is called in the start

///method which is never called

///Demo-> Instantiate a Game object in the Launcher class and

///call the start() method of that object

----->4<-----

Buffers and Drawing

----->4.1<------

Create the render method

- Create a getter for the canvas field

- Declare 2 variables BufferStrategy and Graphics

- Instantiate the BufferStrategy in the render method()

/\*Buffer strategy is a way of telling the computer how to

render graphics on the screen. It uses buffers to do that.

A buffer is a block of memory that help us visualize data

as a block and not consequently. E.g. Draw a head, draw a body

draw hands and legs, put everything as a whole on the screen\*/

- Instantiate the Graphics object

- Call the buffer's show() method and graphic's dispose() method

///Demo-> Fill a rectangle with full width and height using

///the graphics object and run the application. The screen

///should be gray

------>5<-------

Rectangles and Images

------>5.1<-----

- Explain Canvas' coordinate system

- Play arround with drawRect, fillRect, setColor methods

in the graphics

- Overlap rectangles, put negative coordinates etc.

------>5.2<-----

- Remove the rectangles

- Create new folder for the images

- In order to access images we shoudl ake the folder a

resource type of folder

/\*Click the project->properties. Go to the Build Path and

click Add Class Folder\*/

------>5.3<-----

Create an ImageLoader class

/\*Images will be saved in a buffered image object\*/

- Create a pulic static BufferedImage method() called loadImage

it returns a buffered image in a specified path

///Demo-> Test out the ImageLoader class

///Create a temporary BufferedImage field

///Set initialize it in the init() method with the path of

///the file (knowing that the it starts with /) for example

///"/textures/bckg.jpg" this will be an image stored in

///a folder named "textures" inside the images folder

///Call the graphics object in the render() method and

///set it's parameters to {imageVariable, width, height, null}

------->5.4<-----

Use a Sprite Sheet

- Explain what a Sprite Sheet is

///Demo-> Load a SpriteSheet in the game

///Use the same way as last time it should show the whole sprite

- Create a Sprite Sheet class

- Create a Constructor

- Create a crop() method

///Demo-> Load a new SpriteSheet image

///Call the drawImage method with

///g.drawImage(spriteSheet.crop(0,0,95,130), 5, 5, null) params

------->5.5<------

Assets

/\*Any type of texture/image/sound\*/

- Delete the test code

- Create an init() method that will load everything

- Create a static BufferedImage field for every image that you

use

- Initialize every image as cropped from the SpriteSheet

------->6<-------

Fixing the tick() method to run correctly

/\*Limit the tick() and render() metod. How many times they are

called every second to assure that the game runs correctly

on slow and on fast computers the same speed\*/

- Create the fields [fps, timePerTick, delta, now, lastTime]

- Initialize them in the while-loop

- Add an if-condition to the loop

\*- If we want we can add a timer that will check if we are

rendering in 60 fps

------->7<-------

States

/\*Game states are the different stages or frames that should

appear. E.g. Main menu, settings, the game itself are three

different states.\*/

- Create an abstract class called State that will hold everything

that every state has in common

- Declare a tick() and render() abstract methods

- The render() method takes a graphics object so that he can

write directly to the canvas

- Create a GameState class extending the State class

- Create a StateManager by creating a currentState static

field with a getter and a setter

/\*It will hold the current state of the game\*/

///Demo->Check if states work correctly

///Create a MenuState class

///Create a SettingsState class

///Be sure to always set the currentState to game state as we

///do not have any more options yet

-------->8<--------

Creating a player

- Create a Player class

- Create the players fields and constructor

- Create an Intersects() method

- Create a tick() method

- Create a render() method

///Demo-> Test out that the player is visible

------->9<--------

Creating Input Handler

- Create an InputHandler class implementing KeyListener

- Implement unimplemented methods

- Create constructor with Display as parameter

- call display.getCanvas().addKeyListener() method

- Add the player movemeng/shooting logic in the unimplemented methods

///Demo-> Test the InputHandler

///Initialize it in the init() method of the game and give

///a this.display argument

///Initialize a player and an enemy(static non-moving rectangle)

///Write an if-condition if the player intersects the enemy

///Console.log("you died") to check the intersection and

///player movement