PART THREE – CREATING MISSILES

The game is now ready to add in functionality that will fire a missile. This will use some of the skills you already have (creating classes and adding methods, using return values, adding images to a layout). You will also make use of iteration and arrays.

SETTING UP THE BASIC MISSILE

Create a Missile class and add getImage() and move() methods. Remember that the move is in the **y axis**. Create a field (a public, global variable) called STILL_FIRING. Set this as false when the missile is created.

This is my basic Missile class:

```
public class Missile {
    public boolean STILL_FIRING;
    int x, y, dy;
    ImageView ivView;

    Missile(){
        STILL_FIRING = false;
        dy=10;
        y = 400;
        Image imgMissile = new
            Image(getClass().getResourceAsStream("missile.png"));
        ivView = new ImageView(imgMissile);
        ivView.setLayoutY(y);
    }

    public ImageView getImageView(){
        return ivView;
    }

    public void move(){
        y = y-dy;
        ivView.setLayoutY(y);
        if(y<0){
           STILL_FIRING = false;
        }
    }

}
```

The constructor method initialises the speed of the missile (dy) and the starting y position. After creating an Image and initialising the ImageView, y is used to set the start y position of the missile so it lines up with the rocket.

POSITIONING AND FIRING THE MISSILE

The problem so far is that we don't know what the x position of the rocket should be. This means that we can't line it up with the rocket. In order to solve this problem, we can add a getX() to the rocket that will return the current x position. We can then pass this to the missile via a fire() method.

Here is my method to get the current x position of the rocket sprite:

```
public int getX(){
    return x;
}
```

This is the method to set the fire the missile by setting the current x position:

```
public void fire(int rocketX){
    x = rocketX;
    ivView.setLayoutX(x);
    STILL_FIRING = true;
}
```

Before I can fire the missile I need to make sure an instance of it is created in the main class. In the screenshot below you can see (underlined in orange) I have created a global object for the missile, and instantiated it with the rocket and baddie.

```
public class RetroArcadeGame extends Application{
Rocket rocket;
Baddie baddie;
Missile missile;

    @Override
    public void start(Stage primaryStage){
        Group g = new Group();
        Scene scene = new Scene(g, 800,600);

        rocket = new Rocket();
        baddie = new Baddie();
        missile = new Missile();
```

The next step is to edit the action listener to act on the fire command. The original main class had prepared a case for the press of the spacebar which had not been implemented yet.

```
case SPACE:
    missile.fire(rocket.getX());
    g.getChildren().add(missile.getImageView());
    break;
```

This implementation calls the fire method, and passes the x position of the rocket as an argument.

If you run your game now, you should see the missile appear next to the rocket when you press space.

The next step is to move the missile up the screen toward the baddie. To do this we need to add some code to our timer that calls the move method with each frame.

```
new AnimationTimer(){
        @Override
        public void handle(long now){
            baddie.move();
            if (missile.STILL_FIRING){
                missile.move();
            }
        }
}.start();
```

This is my amended AnimationTimer. It checks to see if the missile has been fired (because still firing would be set to true by the fire() method) and then moves the missile. Once the missile hits the top of the screen the move() method in Missile will set STILL_FIRING to false.

Run your game and see what happens. You should be able to move the rocket and fire a missile to the top of the screen. Experiment with removing the missile from the root layout (Group) when it is no longer firing.

MULTIPLE MISSILES

You may have noticed that this is not very effective, what if we want to fire multiple missiles? We want to be able to instantiate any number of missiles as needed, and then remove them from screen once they are no longer needed.

We could create an array of missiles, however in Java an array is fixed in its dimensions. What we want is a data structure that will grow based on the number of missiles that we fire/stop firing. This problem can be solved by using an ArrayList.

The ArrayList class allows us to create a dynamic data structure that will grow depending on our needs. One further advantage is that once we remove the spent missile from our ArrayList, the garbage cleaner will tidy up for us by removing the object and clearing the memory.

This is going to involve completing the following steps:

1. Change the missile constructor to be passed a value for startXpos on instantiation
2. Initialise STILL_FIRING as true; this is because the missile is created only when it needs to be fired
3. Remove the global missile in our main class and replace it with a global ArrayList
4. Remove the fire() method; change the ActionListener so that on space, we will add a new missile to the ArrayList and add the returned ImageView of the new missile to the root layout (Group) so the missile is visible on screen
5. Add a for loop to the AnimationTimer to move each missile in our ArrayList

1 CHANGING THE MISSILE CONSTRUCTOR

```
Missile(int rocketX){
    x = rocketX;
```

Here you can see my modified constructor method now has a parameter that requires an integer representing the x position of the rocket to be passed to it. The x is initialised to the value of this parameter.

2 STILL FIRING

```
Missile(int rocketX){
    x = rocketX;
    STILL_FIRING = true;
    dy = 10;
    y = 400;
    Image imgMissile = new Imag
    ivView = new ImageView(imgM
    ivView.setLayoutY(y);
    ivView.setLayoutX(x);
}
```

Here you can see I have continued to change my constructor method by altering the initial value of still firing. I have also set the X position of the ImageView representing the missile.

3 CREATING AN ARRAYLIST

```
import java.util.ArrayList;
/**
 *
 * @author Yatish
 */
public class RetroArcadeGame extends Application{
Rocket rocket;
Baddie baddie;
ArrayList missileBank;
```

Here I am have imported the package to create an ArrayList into my main class. (Orange underline.) I have also deleted my global Missile, and replaced it with a global ArrayList. (Purple underline.)

In my main method I have replaced the creation of the missile with the new instance of this global ArrayList.

```
rocket = new Rocket();
baddie = new Baddie();
missileBank = new ArrayList();
```

4 MODIFYING THE FIRE() METHOD AND THE ACTIONLISTENER

When the ActionListener is triggered, it will create an instance of Missile that will already be in a firing state. This means the fire() method is technically not needed. I have deleted if from my class.

Next, I need to modify the ActionListener for SPACE.

```
case SPACE:
    missileBank.add(new Missile(rocket.getX()));
    g.getChildren().add(((Missile)missileBank.get((missileBank.size()-1))).getImageView());

    break;
```

This amendment adds a new Missile to the missileBank using the x position of the rocket sprite.

It then:

- Retrieves the last element in the ArrayList missileBank by subtracting one from the size (orange underline)
- Typecasts the retrieved element using the Missile class (blue underline)
- Accesses the ImageView using the getImageView() method (green underline).

5 LOOPING TO MOVE MISSILES

Finally, we can change the AnimationTimer to now move each missile in the ArrayList with each frame.

```
new AnimationTimer(){
        @Override
        public void handle(long now){
            baddie.move();
            for (Object m1:missileBank){
            Missile m  = (Missile)m1;
            m.move();
            }
        }
}.start();
```

This loops through the array list. A Missile m is typecasted for each object m1 that is retrieved from the array. Finally the move method is called for this missile.
If you want to improve the appearance of your code, the loop could be moved into its own method which is called from the AnimationTimer.

REMOVING MISSILES

Missiles are currently being created but not destroyed. We need to add a method that will update the missileBank. If you go back and check your move() method for the missile you will notice that STILL_FIRING is set to false when it goes over the top edge of the screen.

```
new AnimationTimer(){
        @Override
        public void handle(long now){
            baddie.move();
            for (Object m1:missileBank){
            Missile m  = (Missile)m1;
            m.move();
            updateMissiles();
            }
        }
```

Here I have added a line that calls a an updateMissiles() method. I have added this method to my main class.

This is my updateMissiles() method:

```
void updateMissiles(){
for(int i = 0; i<missileBank.size();i++){
    Missile m = (Missile)missileBank.get(i);
    if (!m.STILL_FIRING){
        g.getChildren().remove(i+3);
        missileBank.remove(i);
    }
}
}
```

It loops through the ArrayList. Each element is typecast as a missile and then tested to see if STILL_FIRING is false. If this condition is true then it is removed from the group and the missileBank.