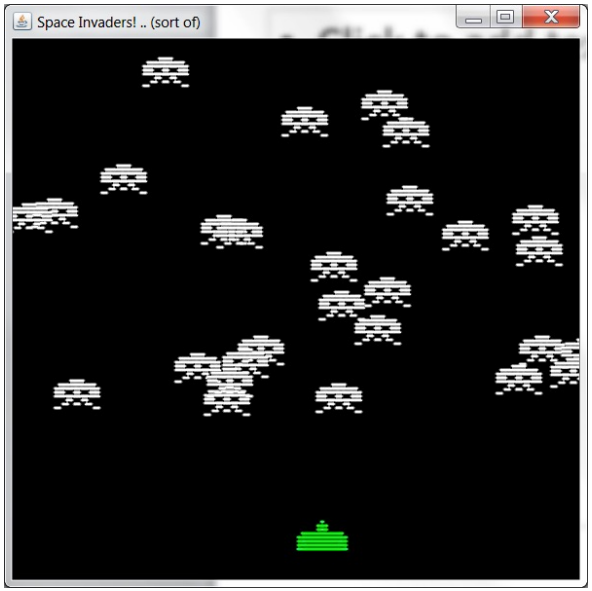
CT255 / NGT2
2D games using Java

Week 4

Sam Redfern
sam.redfern@universityofgalway.ie

Last week's exercise

- Create a JFrame-based, Runnable KeyListener application class and a separate class for handling game objects
- Use these names for your classes:
  - InvadersApplication
  - Sprite2D
- The InvadersApplication class should have an array of Sprite2D objects for aliens, another single Sprite2D object for the player ship, and should use Thread-based animation to move them all (similar to last week)
- The Sprite2D objects display a raster image that you have loaded from disk (instead of a coloured square)
- Use the left and right arrow keys to move the player spaceship, rather than moving it randomly like the aliens



Screen Flicker

- Caused by software redrawing a screen out-of-sync with the screen being refreshed by the graphics hardware (so occasionally a half-drawn image is displayed)
- Solution: use ‘double-buffering’:
  - Render all graphics to an offscreen memory buffer
  - When finished drawing a frame of animation, flip the offscreen buffer onscreen during the ‘vertical sync’ period
- Java awt provides a BufferStrategy class which applies the best approach based on your computer’s capabilities

Implementing Double Buffering

- **In the imports section at the top of the program:**

```
import java.awt.image.*;
```

- **Add a new member variable to the Application class:**

```
private BufferStrategy strategy;
```

- **In the Application class' constructor function:**

```
createBufferStrategy(2);
strategy = getBufferStrategy();
```

- **NB this code should be executed *after* the JFrame has been displayed, i.e. after setBounds() and setVisible().. why might that be?**

- **At the start of the paint(Graphics g) method (redirect our drawing calls to the offscreen buffer):**

```
g = strategy.getDrawGraphics();
```

- **At the end of the paint(Graphics g) method (indicate that we want to flip the buffers):**

```
strategy.show();
```

Let's consider some refactoring..

Sprite2D

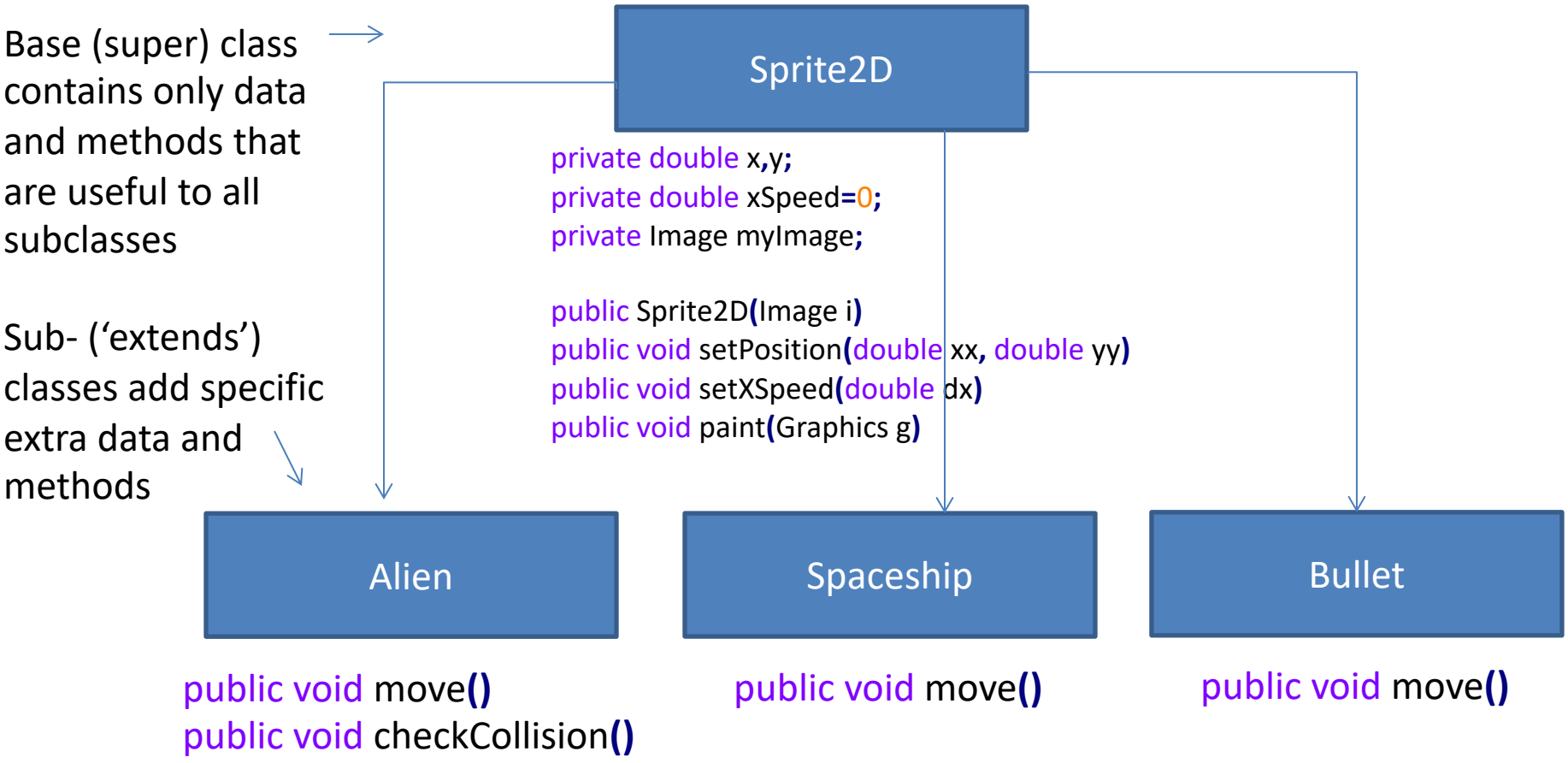
```
private double x,y;
private double xSpeed=0;
private Image myImage;

public Sprite2D(Image i)
public void moveEnemy()
public void setPosition(double xx, double yy)
public void movePlayer()
public void setXSpeed(double dx)
public void paint(Graphics g)
```

We're currently using one class to handle both Aliens and the PlayerShip objects

Some member variables and methods are used by both types of objects, while others are specific to one or the other

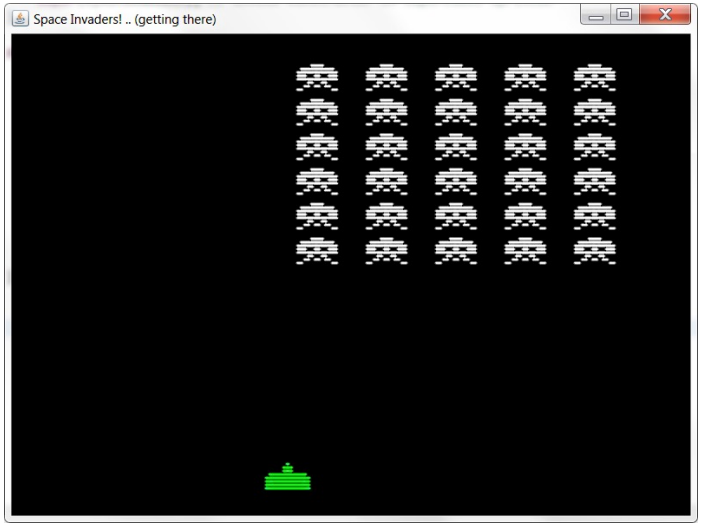
What about when we add Bullets as a third type of object? -> the Sprite2D class gets bloated, confusing and inefficient

Let's consider some refactoring..



This week's assignment

- We're moving closer to a finished game!
- Refactor the game:
  - Make the application window larger (800x600)?
  - Create an **Alien** class and a **Spaceship** class, both subclasses of **Sprite2D**. Move functionality from Sprite2D to the new classes as appropriate.
  - Modify the member variables of the **InvadersApplication** class so that it stores an array of **Alien** objects and a single **Spaceship** object (previously all were Sprite2D objects)
  - Make sure all of the above is working before moving on!



- Implement double buffering to get rid of flickering
- Initialise the aliens in a grid formation rather than randomly positioned
- Modify alien movement so that they all move left or right together (i.e. aliens should use the xSpeed variable similar to how the spaceship does)
- Make all the aliens reverse their movement direction and move down a bit when *any* of them hits the edge of the screen.. ***but how?***

Some useful points regarding class inheritance

1. Rather than using ‘private’ members in the superclass, declare them as ‘protected’ in order for the subclass to be able to access them
2. To call the constructor from a base class, use super();

- E.g:

```
public class Spaceship extends Sprite2D {
        public Spaceship(Image i) {
                super(i); // invoke constructor on superclass (Sprite2D)

                // and do any Spaceship-specific initialisation here..

        }
}
```

Suggested class interfaces

InvadersApplication.java

```
import java.awt.*;

public class InvadersApplication extends JFrame implements Runnable, KeyListener {

    // member data
    private static final Dimension WindowSize = new Dimension(800,600);
    private BufferStrategy strategy;
    private static final int NUMALIENS = 30;
    private Alien[] AliensArray = new Alien[NUMALIENS];
    private Spaceship PlayerShip;

    // constructor
    public InvadersApplication() {

    // thread's entry point
    public void run() {

    // Three Keyboard Event-Handler functions
    public void keyPressed(KeyEvent e) {

    public void keyReleased(KeyEvent e) {

    public void keyTyped(KeyEvent e) {
    }
    //

    // application's paint method
    public void paint(Graphics g) {

    // application entry point
    public static void main(String[] args) {

}
```

Sprite2D.java

```
import java.awt.*;

public class Sprite2D {

    // member data
    protected double x,y;
    protected double xSpeed=0;
    protected Image myImage;
    int winWidth;

    // constructor
    public Sprite2D(Image i, int windowWidth) {

    public void setPosition(double xx, double yy) {

    public void setXSpeed(double dx) {

    public void paint(Graphics g) {

}
```

Alien.java

```
import java.awt.Image;

public class Alien extends Sprite2D {

    public Alien(Image i, int windowWidth) {

    // public interface
    public boolean move() {

    public void reverseDirection() {

}
```

Spaceship.java

```
import java.awt.Image;

public class Spaceship extends Sprite2D {

    public Spaceship(Image i, int windowWidth) {

    public void move() {

}
```