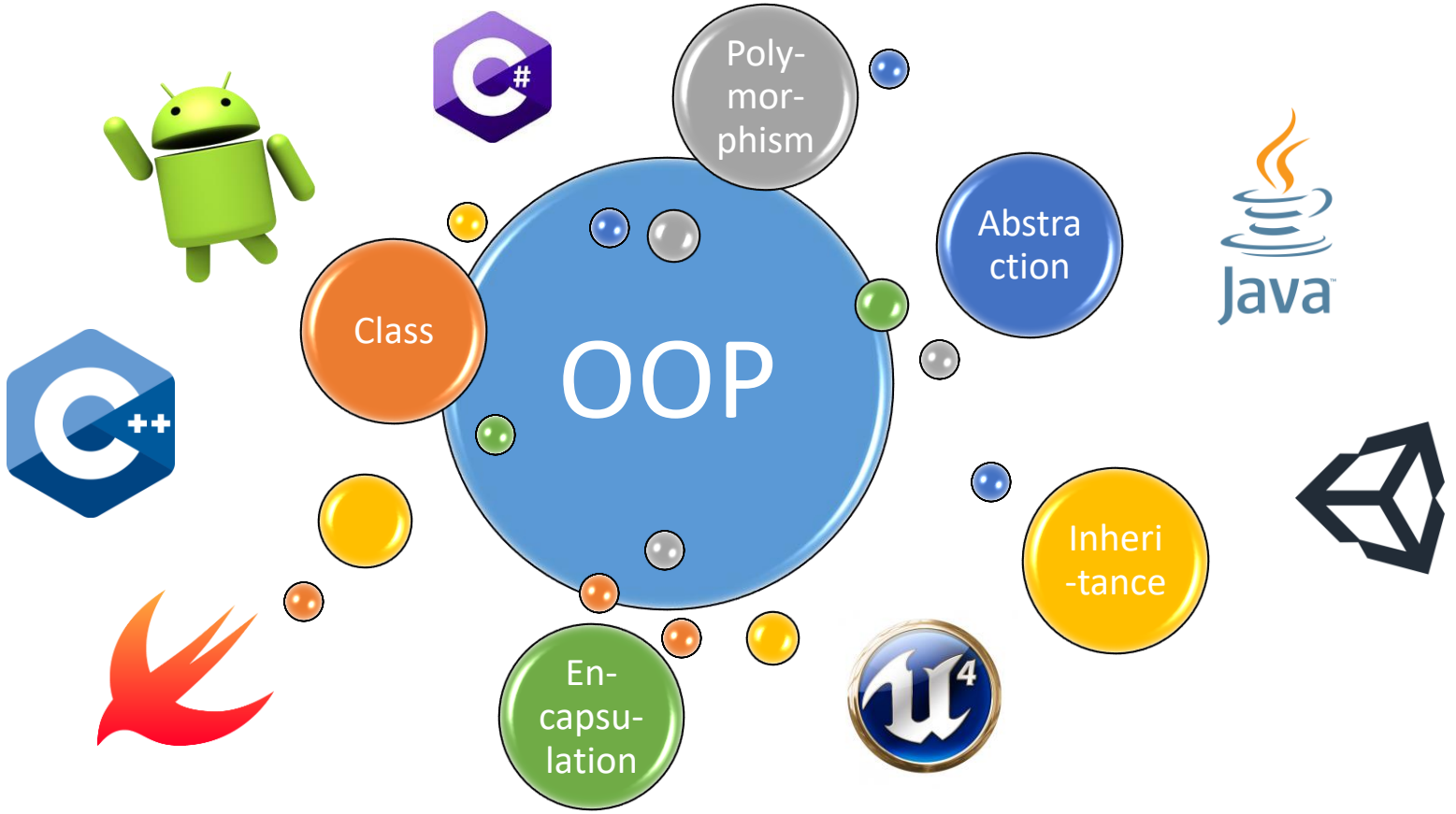


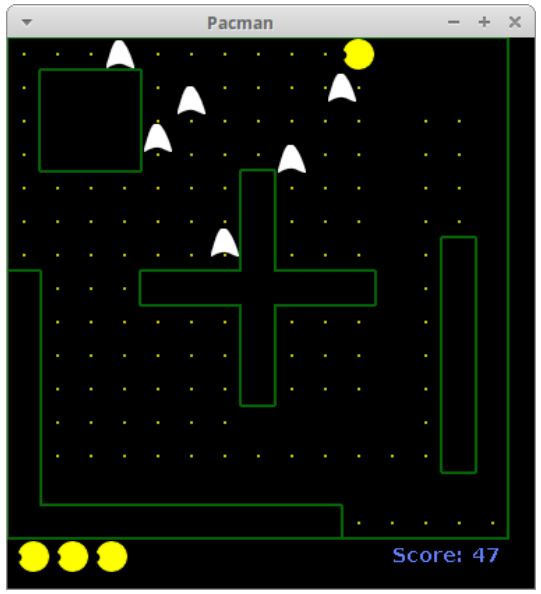
Java 2D Game

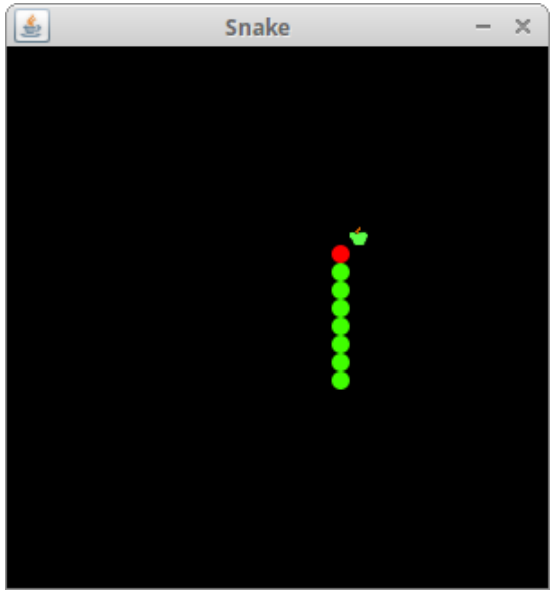
Kuan-Ting Lai

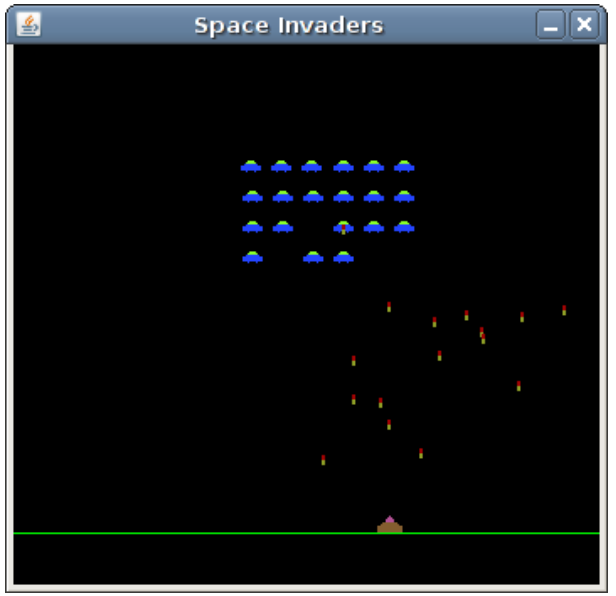
2019/4/25

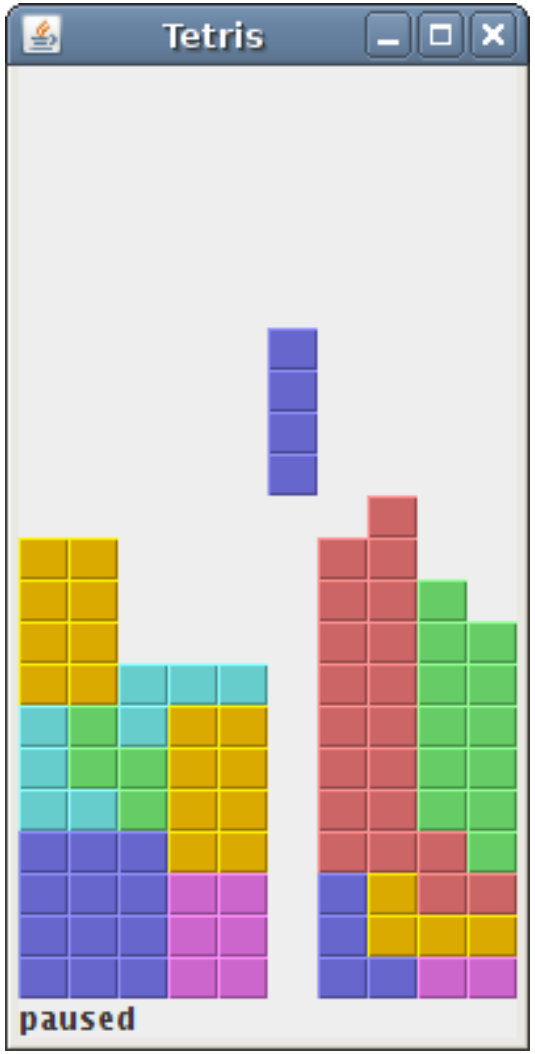
Java 2D Game Tutorial

- http://zetcode.com/tutorials/javagamestutorial/









Running Examples from Zetcode

- Download images
  - http://zetcode.com/img/gfx/javagames/images.zip

- The example codes are packaged by the command
  - package com.zetcode;
- To compile and run the code, you need to
  1. Build folder “com/zetcode”
  2. Put the source codes in the folder above
  3. javac com/zetcode/*.java
  4. java com.zetcode.ImageExample

Using JPanel to Display Screen

- Create class Board, which inherits from JPanel

```
import javax.swing.JPanel;

public class Board extends JPanel {

    public Board() {}
}
```

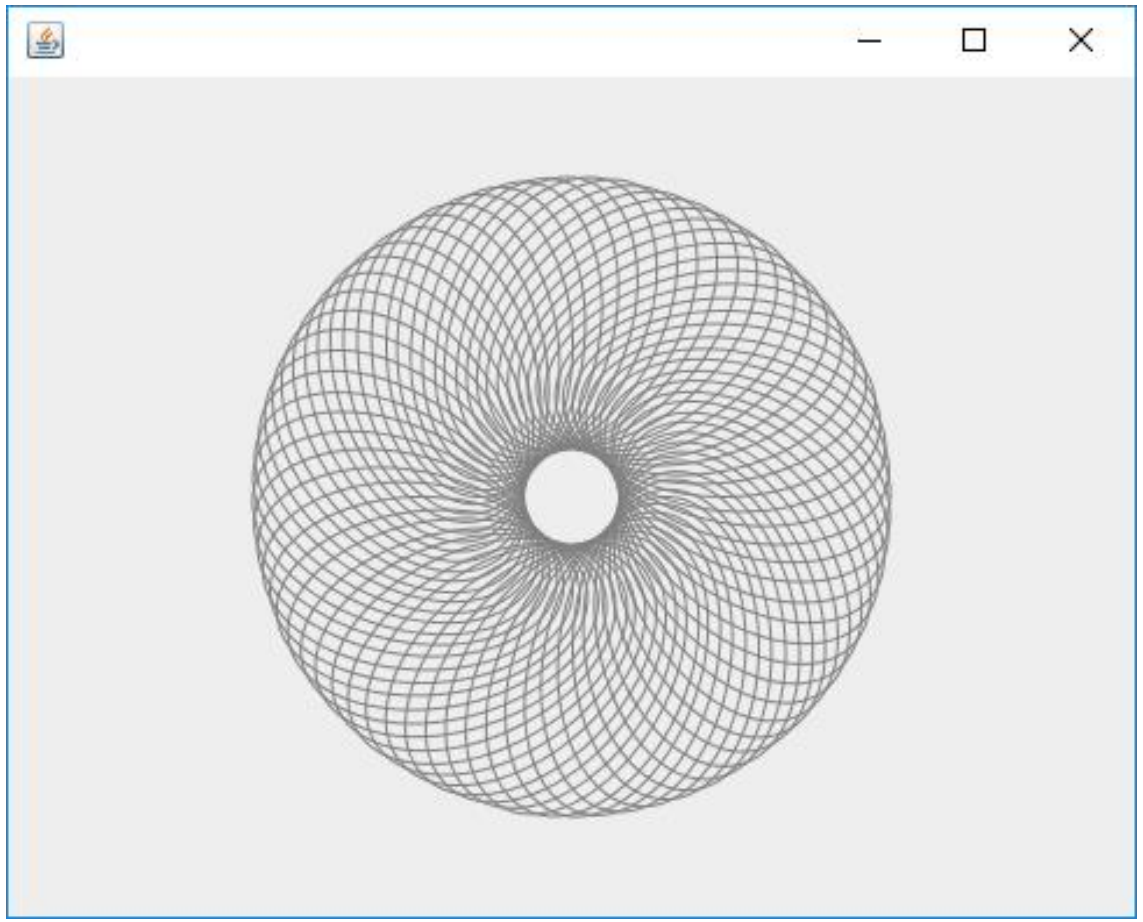
Creating main()

- Using EventQueue
- Swing processing is done in a thread called **EDT (Event Dispatching Thread)**
- EventQueue.*invokeLater*() posts an event (your Runnable) at the end of Swings event list and is processed after all previous GUI events are processed.

```
import java.awt.EventQueue;
import javax.swing.JFrame;

public class Application extends JFrame {

    public Application() {
        add(new Board());

        setSize(250, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
    }


    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            Application ex = new Application();
            ex.setVisible(true);
        });
    }
}
```

https://stackoverflow.com/questions/22534356/java-awt-eventqueue-invokelater-explained

Drawing a Donut



Updating Board.java

- Override `paintComponent(Graphics g)`

```
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.AffineTransform;
import java.awt.geom.Ellipse2D;
import javax.swing.JPanel;

public class Board extends JPanel {

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        drawDonut(g);
    }

    private void drawDonut(Graphics g) {…}
}
```

drawDonut(Graphics g)

```
private void drawDonut(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

    Dimension size = getSize();
    double w = size.getWidth();
    double h = size.getHeight();

    Ellipse2D e = new Ellipse2D.Double(0, 0, 80, 130);
    g2d.setStroke(new BasicStroke(1));
    g2d.setColor(Color.gray);

    for (double deg = 0; deg < 360; deg += 5) {
        AffineTransform at = AffineTransform.getTranslateInstance(w/2, h/2);
        at.rotate(Math.toRadians(deg));
        g2d.draw(at.createTransformedShape(e));
    }
}
```

Drawing an Image

```java
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JPanel;

public class Board extends JPanel {
    private Image bardejov;
    public Board() {
        initBoard();
    }
    private void initBoard() {
        loadImage();
        int w = bardejov.getWidth(this);
        int h =  bardejov.getHeight(this);
        setPreferredSize(new Dimension(w, h));
    }
    private void loadImage() {
        ImageIcon ii = new ImageIcon("src/resources/beach-walking-family.jpg");
        bardejov = ii.getImage();
    }
    @Override
    public void paintComponent(Graphics g) {
        g.drawImage(bardejov, 0, 0, null);
    }
}
```

Animation

- http://zetcode.com/tutorials/javagamestutorial/animation/
- Showing a moving star

Board.java (2-1)

- **implements** ActionListener
- Create a timer
- @Override

void
actionPerformed(ActionEvent e)

```
public class Board extends JPanel implements ActionListener {
    private final int B_WIDTH = 350;
    private final int B_HEIGHT = 350;
    private final int INITIAL_X = -40;
    private final int INITIAL_Y = -40;
    private final int DELAY = 25;

    private Image star;
    private Timer timer;
    private int x, y;

    public Board() {
        initBoard();
    }
    private void loadImage() {
        ImageIcon ii = new ImageIcon("src/resources/star.png");
        star = ii.getImage();
    }
    private void initBoard() {…}
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        drawStar(g);
    }
    private void drawStar(Graphics g) {…}

    @Override
    public void actionPerformed(ActionEvent e) {…}
}
```

Board.java (2-2)

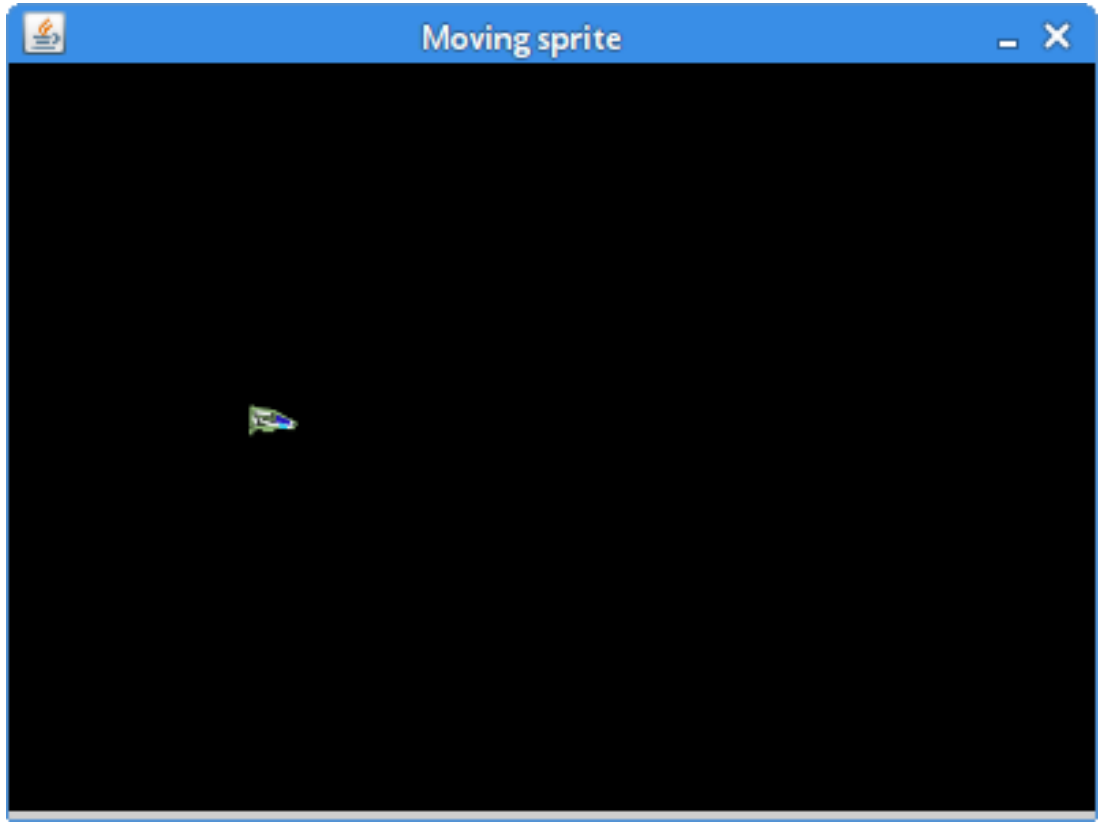
- Call `drawStar(…) in paintComponent(…)`
- Call repaint() to refresh the display

```
public class Board extends JPanel implements ActionListener {
    ……
    public Board() {…}
    private void loadImage() {…}
    private void initBoard() {
        setBackground(Color.BLACK);
        setPreferredSize(new Dimension(B_WIDTH, B_HEIGHT));
        loadImage();
        x = INITIAL_X;
        y = INITIAL_Y;
        timer = new Timer(DELAY, this);
        timer.start();
    }
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawImage(star, x, y, this);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        x += 1;
        y += 1;
        if (y > B_HEIGHT) {
            y = INITIAL_Y;
            x = INITIAL_X;
        }
        repaint();
    }
}
```

Moving Sprites

Moving Sprite Example

- http://zetcode.com/tutorials/javagamestutorial/movingsprites/



SpaceShip

- Three important functions
  - move()
  - keyPressed(KeyEvent e)
  - keyReleased(KeyEvent e)

```
import ...
public class SpaceShip {
    public int x = 40;
    public int y = 60;
    public int w;
    public int h;
    private int dx;
    private int dy;
    private Image image;

    public SpaceShip() { loadImage();}
    private void loadImage() {…}
    public Image getImage() { return image;}
    public void move() {
        x += dx;
        y += dy;
    }
    public void keyPressed(KeyEvent e) {
        int key = e.getKeyCode();
        switch(key) {
            case KeyEvent.VK_LEFT:  dx = -2; break;
            case KeyEvent.VK_RIGHT: dx = 2; break;
            case KeyEvent.VK_UP:    dy = -2; break;
            case KeyEvent.VK_DOWN:  dy = 2; break;
        }
    }
    public void keyReleased(KeyEvent e) {
        int key = e.getKeyCode();
        switch(key) {
            case KeyEvent.VK_LEFT:  dx = 0; break;
            case KeyEvent.VK_RIGHT: dx = 0; break;
            case KeyEvent.VK_UP:    dy = 0; break;
            case KeyEvent.VK_DOWN:  dy = 0; break;
        }
    }
}
```

Inheriting **KeyAdapter**

- Create a new private class TAdapter, which extends KeyAdaptor to ovewrite keyReleased and keyPressed events

```
private class TAdapter extends KeyAdapter {

    @Override
    public void keyReleased(KeyEvent e) {
        spaceShip.keyReleased(e);
    }

    @Override
    public void keyPressed(KeyEvent e) {
        spaceShip.keyPressed(e);
    }
}
```

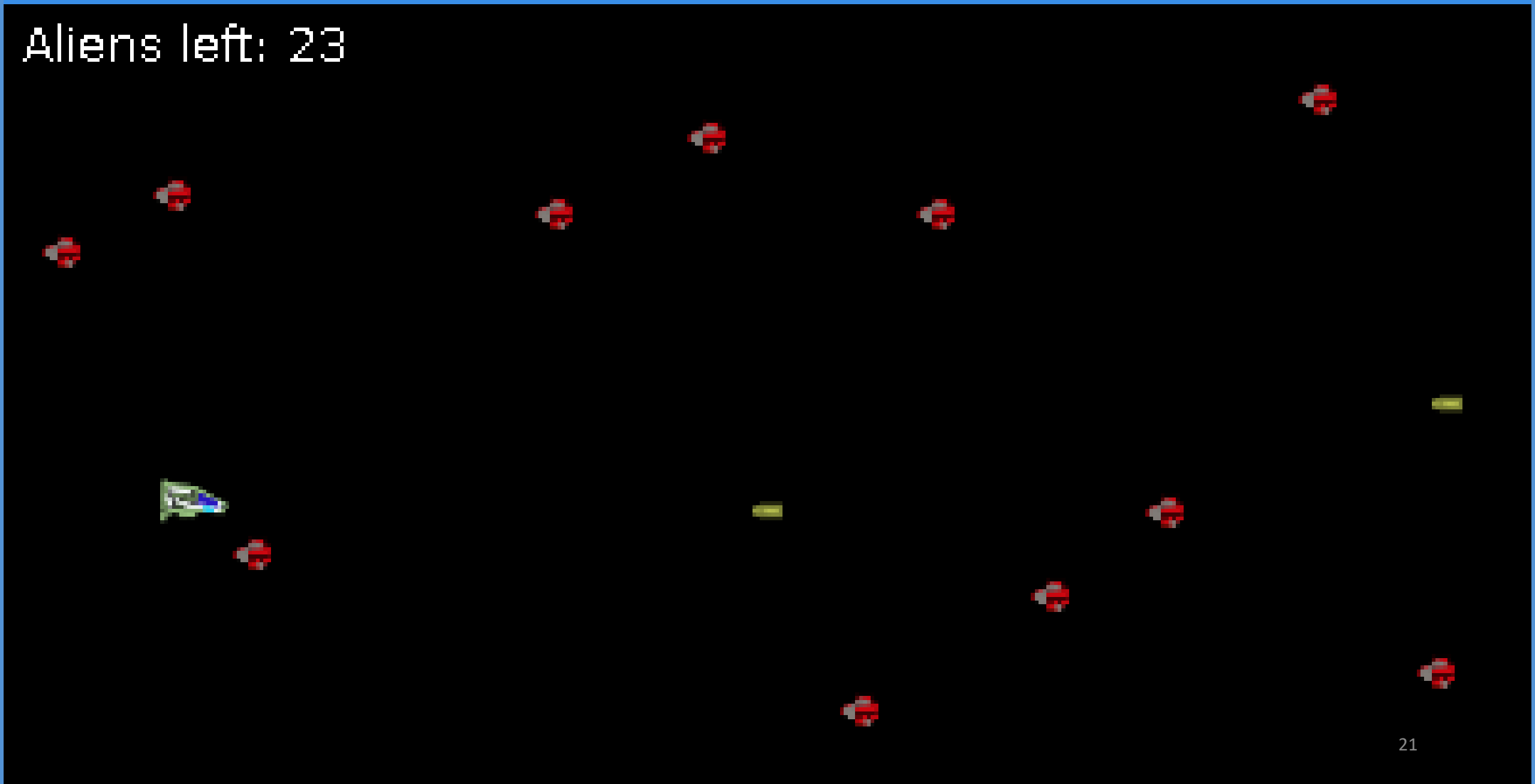
Board.java

```
public class Board extends JPanel implements ActionListener
{
    private Timer timer;
    private SpaceShip spaceShip;
    private final int DELAY = 10;
    public Board() {
        addKeyListener(new TAdapter());
        setBackground(Color.black);
        setFocusable(true);
        spaceShip = new SpaceShip();
        timer = new Timer(DELAY, this);
        timer.start();
    }
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawImage(spaceShip.getImage(), spaceShip.x, spaceShip.y, this);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        step();
    }
    private void step() {
        spaceShip.move();
        repaint(spaceShip.x-1, spaceShip.y-1, spaceShip.w+2, spaceShip.h+2);
    }
    private class TAdapter extends KeyAdapter {……}
}
```

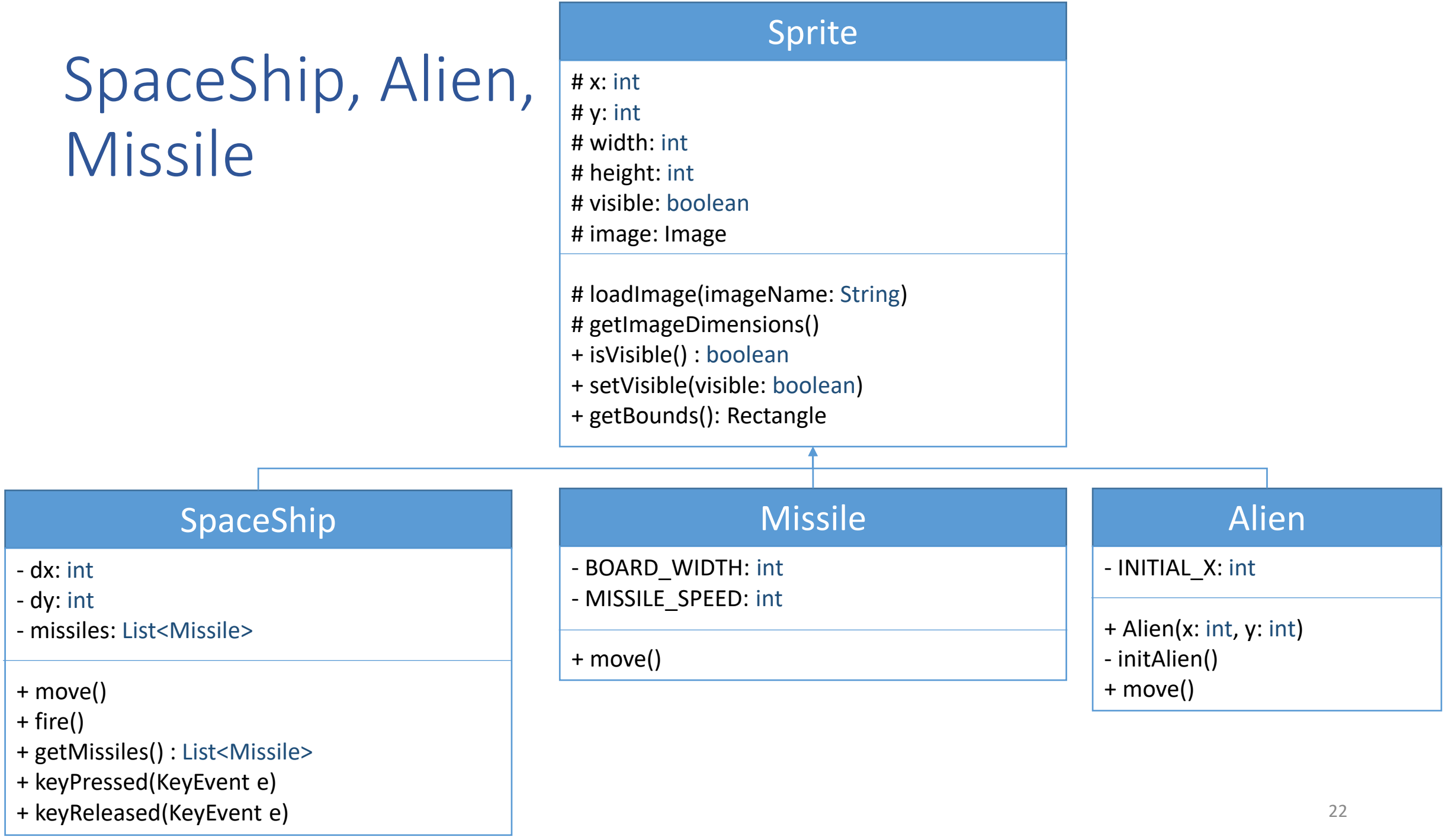
MovingSpriteEx

```
public class MovingSpriteEx extends JFrame {
    public MovingSpriteEx() {
        initUI();
    }
    private void initUI() {
        add(new Board());
        setTitle("Moving sprite");
        setSize(800, 600);

        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String[] args) {
        EventQueue.invokeLater(() -> {
            MovingSpriteEx ex = new MovingSpriteEx();
            ex.setVisible(true);
        });
    }
}
```

Collision Detection



SpaceShip, Alien, Missile



SpaceShip

- Add list of missiles
  - List<Missile> missiles
- Fire missiles
  - `missiles.add(new Missile(x + width, y + height / 2));`

```
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;

public class SpaceShip extends Sprite {
    private int dx;
    private int dy;
    private List<Missile> missiles;

    public SpaceShip(int x, int y) {
        super(x, y);
        initCraft();
    }
    private void initCraft() {
        missiles = new ArrayList<>();
        loadImage("src/resources/craft.png");
        getImageDimensions();
    }
    public List<Missile> getMissiles() {
        return missiles;
    }
    public void fire() {
        missiles.add(new Missile(x + width, y + height / 2));
    }

    public void move() {...}
    public void keyPressed(KeyEvent e) {...}
    public void keyReleased(KeyEvent e) {...}
}
```

Missile

- Set default speed
  - MISSILE_SPEED = 2
- Move the missile

```
public class Missile extends Sprite {
    private final int BOARD_WIDTH = 390;
    private final int MISSILE_SPEED = 2;

    public Missile(int x, int y) {
        super(x, y);
        initMissile();
    }

    private void initMissile() {
        loadImage("src/resources/missile.png");
        getImageDimensions();
    }

    public void move() {

        x += MISSILE_SPEED;

        if (x > BOARD_WIDTH)
            visible = false;
    }
}
```

Alien

- Moving toward left constantly

```
public class Alien extends Sprite {
    private final int INITIAL_X = 400;

    public Alien(int x, int y) {
        super(x, y);
        initAlien();
    }

    private void initAlien() {
        loadImage("src/resources/alien.png");
        getImageDimensions();
    }

    public void move() {

        if (x < 0) {
            x = INITIAL_X;
        }
        x -= 1;
    }
}
```

Board

- Initialize SpaceShip, Aliens
- Draw objects
  - updateShip()
  - updateMissiles()
  - updateAliens()
  - drawObjects()
- Check collisions
  - checkCollisions()
- Check gameover
  - inGame()
  - drawGameOver

Board
- timer: Timer - spaceship: SpaceShip - ingame: Boolean - ICRAFT_X: int - ICRAFT_Y: int - DELAY: int - alien_pos: int[][] - TAdaptor: KeyAdaptor
+ Board() + initAlients() + paintComponent(Graphics g) + drawObjects(Graphics g) + drawGameOver(Graphics g) + actionPerformed(Action e) + inGame() + updateShip() + updateMissiles() + updateAliens() + checkCollisions

Initializing Parameters

- Assign aliens' positions
  - int[][] pos

```
public class Board extends JPanel implements
ActionListener {

    private Timer timer;
    private SpaceShip spaceship;
    private List<Alien> aliens;
    private boolean ingame;
    private final int ICRAFT_X = 40;
    private final int ICRAFT_Y = 60;
    private final int B_WIDTH = 400;
    private final int B_HEIGHT = 300;
    private final int DELAY = 15;

    private final int[][] pos = {
            {2380, 29}, {2500, 59}, {1380, 89},
            {780, 109}, {580, 139}, {680, 239},
            {790, 259}, {760, 50}, {790, 150},
            {980, 209}, {560, 45}, {510, 70},
            {930, 159}, {590, 80}, {530, 60},
            {940, 59}, {990, 30}, {920, 200},
            {900, 259}, {660, 50}, {540, 90},
            {810, 220}, {860, 20}, {740, 180},
            {820, 128}, {490, 170}, {700, 30}
    };

...
...
}
```

Board Initialization

- Create Timer
- Create SpaceShip and Aliens

```
.....
public Board() {
    initBoard();
}
private void initBoard() {
    addKeyListener(new TAdapter());
    setFocusable(true);
    setBackground(Color.BLACK);
    ingame = true;
    setPreferredSize(new Dimension(B_WIDTH, B_HEIGHT));

    spaceship = new SpaceShip(ICRAFT_X, ICRAFT_Y);
    initAliens();

    timer = new Timer(DELAY, this);
    timer.start();
}
public void initAliens() {
    aliens = new ArrayList<>();
    for (int[] p : pos) {
        aliens.add(new Alien(p[0], p[1]));
    }
}
.....
```

Drawing

```
……
@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    if (ingame) {
        drawObjects(g);
    } else {
        drawGameOver(g);
    }
    Toolkit.getDefaultToolkit().sync();
}
private void drawObjects(Graphics g) {
    if (spaceship.isVisible()) {
        g.drawImage(spaceship.getImage(), spaceship.getX(), spaceship.getY(), this);
    }
    List<Missile> ms = spaceship.getMissiles();
    for (Missile missile : ms) {
        if (missile.isVisible()) {
            g.drawImage(missile.getImage(), missile.getX(), missile.getY(), this);
        }
    }
    for (Alien alien : aliens) {
        if (alien.isVisible()) {
            g.drawImage(alien.getImage(), alien.getX(), alien.getY(), this);
        }
    }
    g.setColor(Color.WHITE);
    g.drawString("Aliens left: " + aliens.size(), 5, 15);
}
……
```

Drawing Game Over

```
    private void drawGameOver(Graphics g) {
        String msg = "Game Over";
        Font small = new Font("Helvetica", Font.BOLD, 14);
        FontMetrics fm = getFontMetrics(small);

        g.setColor(Color.white);
        g.setFont(small);
        g.drawString(msg, (B_WIDTH - fm.stringWidth(msg)) / 2,
                B_HEIGHT / 2);
    }
```

Processing Timer Events

```
@Override
public void actionPerformed(ActionEvent e) {
    inGame();
    updateShip();
    updateMissiles();
    updateAliens();
    checkCollisions();
    repaint();
}
```

Updating Objects

- Update game status, ship, aliens, and missiles

```
private void inGame() {
    if (!ingame) {
        timer.stop();
    }
}
private void updateShip() {
    if (spaceship.isVisible()) {
        spaceship.move();
    }
}
private void updateMissiles() {
    List<Missile> ms = spaceship.getMissiles();
    for (int i = 0; i < ms.size(); i++) {
        Missile m = ms.get(i);
        if (m.isVisible()) {
            m.move();
        } else {
            ms.remove(i);
        }
    }
}
private void updateAliens() {
    if (aliens.isEmpty()) {
        ingame = false;
        return;
    }
    for (int i = 0; i < aliens.size(); i++) {
        Alien a = aliens.get(i);
        if (a.isVisible()) {
            a.move();
        } else {
            aliens.remove(i);
        }
    }
}
```

Check Collisions

- Check ship-alien collision
- Check missile-alien collision

```
public void checkCollisions() {
    Rectangle r3 = spaceship.getBounds();
    for (Alien alien : aliens) {
        Rectangle r2 = alien.getBounds();
        if (r3.intersects(r2)) {
            spaceship.setVisible(false);
            alien.setVisible(false);
            ingame = false;
        }
    }

    List<Missile> ms = spaceship.getMissiles();
    for (Missile m : ms) {
        Rectangle r1 = m.getBounds();
        for (Alien alien : aliens) {
            Rectangle r2 = alien.getBounds();
            if (r1.intersects(r2)) {
                m.setVisible(false);
                alien.setVisible(false);
            }
        }
    }
}
```

Keyboard Events

```
private class TAdapter extends KeyAdapter
{
    @Override
    public void keyReleased(KeyEvent e) {
        spaceship.keyReleased(e);
    }
    @Override
    public void keyPressed(KeyEvent e) {
        spaceship.keyPressed(e);
    }
}
```

Reference

- http://zetcode.com/tutorials/javagamestutorial/