

Java Game Tutorial - Part 1

In this and following tutorials you will learn about Applets, Threads, Graphics and a few other things. By the end of this tutorial you should have the skills to make basic games in Java. For this tutorial you will be making a simple 'space invaders' type game.

I assume that you have basic knowledge of Java as I won't be going into the basic details of how some things work.

First we will be starting by creating an applet and drawing a circle to the applet area.

1. Create a file called 'Game.java'.
2. Open the file.

The next step is to import the necessary packages. For now we will only be requiring 2 packages:

```
import java.applet.*;
import java.awt.*;
```

Now that the importing has been taken care of we will need to set up the Java applet by the following:

```
public class Game extends Applet implements Runnable
{
}
```

This basically gives access to the Applet class and the 'Runnable' makes it so we can implement threads.

The variables come next as we wish to make these ones global:

```
Thread gameThread;
int width=400, height=400, MAX=1;
int currentX[] = new int[MAX];
int currentY[] = new int[MAX];
```

I have decided to use arrays for the X and Y cords now because they will be used at a later stage. It makes it easier to set it up now rather than changing it later.

Next comes the methods. I have included methods that are currently not used at this stage but they are used later.

Start() is used for starting a new thread for the class.

```
public void start()
{
```

```
    Thread gameThread = new Thread(this);
    gameThread.start();
}
```

init() is used for setting the initial values

```
public void init()
{
    currentX[0]=0;
    currentY[0]=0;
}
```

run() is the main method we will use later. It is initialized after a new thread is started.

```
public void run()
{
}
```

paint() calls update().

```
public void paint(Graphics g)
{
    update(g);
}
```

update () is where all the actual drawing is done.

```
public void update(Graphics g)
{
    Graphics2D g2 = (Graphics2D)g;

    // Set the background color.
    g2.setBackground(Color.black);

    // Clear the applet.
    g2.clearRect(0, 0, width, height);

    // Set the drawing color to green.
    g2.setColor(Color.green);

    //(X pos, Y pos, Width, Height)
    g2.fillOval(currentX[0], currentY[0], 20,20);
}
```

* Note this is an applet which means you must run it from a HTML file. The HTML code to run this applet is as follows and I will use the same code throughout this series of tutorials.



As you will see if you compile and run this applet it will draw a green circle on a black background. This is pretty simple and pretty boring but it sets the basis up for doing things later. At this stage you do not even need the Thread but I thought I would put it in now to make it more easy later.

Now I will go into something more interesting but still rather useless. We will now make the circle bounce around the screen.

In this version I have added more global variables:

```
int speed=10; // Speed at which we will move the objects

// Which direction to move the object
int directionX[] = new int[MAX];
int directionY[] = new int[MAX];
```

These variables are used for making sure the applet doesn't go psycho fast on very fast computers and also to make sure it doesn't run to slow on slow computers.

```
long start=0;
long tick_end_time;
long tick_duration;
long sleep_duration;

static final int MIN_SLEEP_TIME = 10;
static final int MAX_FPS = 20;
static final int MAX_MS_PER_FRAME = 1000 / MAX_FPS;

float fps=0;
```

Only two functions are modified in this version. The first modification is a simple one. All it does is sets the value in directionX[0] to 1 which means that it will travel to the left first, and directionY[0] to 0 which means it will travel to the top first:

```
public void init()
{
    currentX[0]=100;
    currentY[0]=0;
```

```
directionX[0]=1;
directionY[0]=0;
}
```

This module as I said earlier this is the main function and as you will notice this one has the biggest amount of code in it now, so this may take a bit to explain. First here is the run() code in full.

```
public void run()
{
    while(true){
        start = System.currentTimeMillis();

        for(int i=0; i < MAX; i++){
            if(directionX[i]==1)
                currentX[i]+=speed;

            if(directionX[i]==0)
                currentX[i]-=speed;

            if(currentX[i] <= 0)
                directionX[i]=1;

            if(currentX[i]+20 >= width)
                directionX[i]=0;

            if(directionY[i]==1)
                currentY[i]+=speed;

            if(directionY[i]==0)
                currentY[i]-=speed;

            if(currentY[i] <= 0)
                directionY[i]=1;

            if(currentY[i]+20 >= height)
                directionY[i]=0;
        }

        repaint();

        tick_end_time = System.currentTimeMillis();
        tick_duration = tick_end_time - start;
        sleep_duration = MAX_MS_PER_FRAME - tick_duration;
```

```

        if (sleep_duration < MIN_SLEEP_TIME)
        {
            sleep_duration = MIN_SLEEP_TIME;
        }
        fps = 1000 / (sleep_duration + tick_duration);

        try{
            Thread.sleep(sleep_duration);
        } catch(InterruptedException e) {}

    }
}

```

Now onto explaining the function. First this function will continually loop. This is what gives our game moving objects.

The next line is:

```
start = System.currentTimeMillis();
```

This line is part of our frame rate calculations. It will set the start time to the system time at the start of the frame.

The next section is a for loop containing the code for calculating the objects position and where to move it to next. The X and Y cords are similar so I will only explain the X cord:

These first two if statements are used for calculating the new position in relation to its current X value:

```
if(directionX[i]==1)
    currentX[i]+=speed;
```

This says if the current object is moving right then add 'speed' to the current position of the object. Basically if the object is at X:0 and the speed is 10, then the new X position will be 10 (0 + 10).

```
if(directionX[i]==0)
    currentX[i]-=speed;
```

This section does the opposite to the previous if statement (moves to left).

These last two detect if the object has hit the side of the applet and if so change the direction:

```
if(currentX[j] <= 0)
    directionX[i]=1;
```

If the current X position is less than or equal to zero then change the direction so it now moves to the right.

```
if(currentX[i]+20 >= width)
    directionX[i]=0;
```

Again this one does similar but it detects if the object has hit the right side of the applet. This one has a small variation in the checking tho. You will notice it has:

```
currentX[i]+20
```

The current X value of the object is at cord 0 of the object (its left most side). This means that X will only be greater than or equal to the width after all of the right hand size has gone out of the applet. So we must add 20 which is the width of the object. Feel free to remove the +20 or change it to higher and smaller values to observe the effects it has.

That concludes the collision detection section.

Next you will see:

```
repaint();
```

Basically all this line does is calls the functions for painting on the screen paint() and update(). I think it does some internal calculations before going to those functions first though.

You are probably thinking this is all a bit much right now, but don't worry, only 2 more sections to go.

```
tick_end_time = System.currentTimeMillis();
tick_duration = tick_end_time - start;
sleep_duration = MAX_MS_PER_FRAME - tick_duration;

if (sleep_duration < MIN_SLEEP_TIME)
{
    sleep_duration = MIN_SLEEP_TIME;
}
fps = 1000 / (sleep_duration + tick_duration);
```

This code is pretty simple and self explanatory so I won't go into details. But basically it works out how long it has taken to draw the current frame. If the system is lagging it will 'sleep_duration' reduce 'sleep_duration' and if it's going to fast it will increase 'sleep_duration'. The sleep time is done in this final section:

```
try{
    Thread.sleep(sleep_duration);
} catch(InterruptedException e) {}
```

This will simply pause the thread for the value in 'sleep_duration'. This is all done to make sure the game runs at its best on all systems.

If you want you can display the current frame rate by placing this line of code in your `paint(Graphics g)` function:

```
g.drawString("FPS: "+fps,1,400);
```

Also be aware that that calculation doesn't always give the correct frame rate. It is simply to regulate the speed of the app.

Well if you compile all that and run it you should get a green circle bouncing around your screen.

You will also notice that it flickers a lot especially if you increase the frame rate. This is because you can actually see the computer drawing the objects to the screen. In the next tutorial I will explain how to do frame buffering which will make your objects run smooth, and more.

Thanks for reading,
Feel free to send any comments.

Java Game Tutorial - Part 2

Welcome back.

In this tutorial I will show you how to implement buffering into your app. First here are some details on how buffering works.

The current app you have flickers. This is because you can actually see the computer drawing the images to the screen. If you use buffering the computer draws it to one area and then only once it has finished drawing the objects it displays it on the screen.

There are different ways of doing buffering, but for now I will stick with the easiest so you can get an idea of how it all works.

If you have trouble understanding that example this way may help. I like to think of buffering like someone drawing on a piece of paper. If you are watching someone draw on a piece of paper you can see every movement the person makes. But if say the person has 2 pieces of paper it could make things better. The person gives you one to look at. While you are looking at that piece of paper the person goes on and draws on the blank one. Once the person has finished drawing that one, the person will switch pages with you, and start drawing again. I know it's a lame way of putting it but it's simple.

Now that all the background is out of the way we will now get to modifying the code from the first tutorial.

First you will need to import:

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

We also have two new variables to add:

```
BufferedImage bufferdImg;  
Graphics2D bufferdImgSurface;
```

Scroll down until you find the function `init()` and add the following code:

```
bufferdImg = (BufferedImage)createImage(width, height);  
bufferdImgSurface = bufferdImg.createGraphics();
```

These two lines of code will set up the area to be drawn to.

The last step is to modify the `update(Graphics g)` function. The code is as follows:

```
public void update(Graphics g)  
{  
    Graphics2D g2 = (Graphics2D)g;  
    // Set the background color.  
    bufferdImgSurface.setBackground(Color.black);  
  
    // Clear the applet.  
    bufferdImgSurface.clearRect(0, 0, width, height);  
  
    bufferdImgSurface.setColor(Color.green);  
    //(X pos, Y pos, Width, Height)  
    bufferdImgSurface.fillOval(currentX[0], currentY[0], 20,20);  
  
    g2.drawImage(bufferdImg, 0, 0, this);  
}
```

As you see we have changed it from drawing to 'g2' to 'bufferdImgSurface', and only at the very end drawing the whole frame to the screen:

```
g2.drawImage(bufferdImg, 0, 0, this);
```

Now it's ready to go. You should now have a reduction in the flickering. As I said it's not the best way to do it but it works and it's easy so it is fine for now.

This next section is to show you how collision detection will be working in the game. You will notice that there is already some collision detection with the circle bouncing around the screen, but we will now expand on this. Please note most of this code will not be needed in our game so you may want to make a copy of your current file. Also you can skip this section if you wish but I recommend at least reading through it.

Again we will start in the variables section. Locate the integer variable called MAX. The current value of this is 1 (one circle). We want to have 2 circles bouncing around the screen so we will change MAX to 2.

Next we need to add two new variables:

```
boolean collided=false;  
float dist;
```

'collided' is only true if the distance between the two points is less than the specified amount.

'dist' is the distance between the two points.

In the 'init()' function add:

```
currentX[1]=0;  
currentY[1]=100;  
  
directionX[1]=0;  
directionY[1]=1;
```

This code is just the same as previous code so it shouldn't need explaining.

This next section of code should go in the 'run()' function just after the two circles have been moved:

```
dist = (int)(Math.sqrt(Math.pow((currentX[0]+20)-(currentX[1]+20),2) +  
Math.pow((currentY[1]+20)-(currentY[1]+20),2)));  
  
if(dist < 20)  
    collided = true;  
else  
    collided = false;
```

The first line of code is the distance formula:

```
sqrt(pow((X1)-(X2),2) + pow((Y1)-(Y2),2)))
```

This formula just calculates the distance between two points when given the X and Y cords.

The next section is just an if statement that says if the distance between the two points is less than 20 then they must be touching so set 'collided' to true.

Just a note. You may notice that in the distance formula I have the current position +20. This is because I am adding the diameter of the circle or you would only get the absolute X/Y cord.

The last thing to add is to the 'update(Graphics g)' function:

```
bufferdImgSurface.fillOval(currentX[1], currentY[1], 20,20);  
  
if(collided==true)  
    bufferdImgSurface.drawString("Collided",10,10);
```

Add those two lines just b4:

```
g2.drawImage(bufferdImg, 0, 0, this);
```

Compile and run.

You should notice that the two circles both bounce around the screen. When the two circles are touching the word "Collided" is displayed in the top left hand corner.

This is one of the simplest methods of collision detection and the method we will be using to detect a collision between the bullets, player and the enemy(s).

It's been a long time but now all the basics are now out of the way and its now time for us to start working on the actual game.

This will require many modifications and additions to the code. Just to give you an idea of the size of the game, it's around 7 pages.

In this game we will be using the mouse for input we must do a few things to set up the mouse for input. First you must import the following package:

```
java.awt.event.*;
```

You must also add to the class line:

```
public class Game extends Applet implements Runnable, MouseMotionListener,  
MouseListener
```

That is all to the mouse section for now. We will be dealing with the mouse listener more throughout the code, but for now we will move onto the variables.

First you can go and delete these variables as they are no longer needed:

```
int directionX[] = new int[MAX];  
int directionY[] = new int[MAX];
```

There are a lot of new variables to add so I am just going to give you the whole list that we will be using to make your's and my life easier. Some of these you will already have, some you won't. Also the comments next to them should explain them pretty well:

```
BufferedImage bufferdImg;  
Graphics2D bufferdImgSurface;
```

```
Thread gameThread;
```

```
int width=400, height=400, MAX=50, speed=10;
```

```

int currentX[] = new int[MAX];
int currentY[] = new int[MAX];

int step=0,      // Number of movements left/right
direction=1,    // Current left/right direction (0=left, 1=right)
shipX=width/2-10, // Current player X position
shipY=height-45, // Current player Y position
mbx=-10,       // The mouse position after mouse down, sets the_
mby=-10,       // enemy bullet position to this.
randomShoot=0, // Used to work out which enemy is shooting
health=50,     // The players health
BNUM=10,      // Number of bullets
playing=0;    // Are is the game playing (0=Playing, 1=Paused, 2=Game Over, 3=Win)

int bX[] = new int[BNUM]; // Bullet X pos.
int bY[] = new int[BNUM]; // Bullet Y pos.

int ebX[] = new int[BNUM]; // Enemy Bullet X pos.
int ebY[] = new int[BNUM]; // Enemy Bullet Y pos.

long start=0, // Frame start time
tick_end_time, // End frame time
tick_duration, // Time taken to display the frame
sleep_duration; // How long to sleep for

static final int MIN_SLEEP_TIME = 10, // Min time to sleep for
MAX_FPS = 20, // Max frame rate.
MAX_MS_PER_FRAME = 1000 / MAX_FPS; // MS per frame

float fps=0, // Current frame rate
dist; // Distance between 2 points

```

The first function in our code is 'start()' this has no changes so lets move to the next one.

Next is 'init()'. As you remember this function sets our initial values. This has a few additions as follows:

This section of code is for drawing a grid of circles 10 by 5.
Set up local integer variables for keeping track of what we have drawn.

```

int row=10, // Current Y position
col=10, // Current X position
count=0; // How many circles have been drawn

```

We will set the first circle to the initial values of 'row' and 'col' so we have a starting point to work from:

```
currentX[0]=col;
currentY[0]=row;
```

This section actually sets the coordinates for each circle:

```
for(int i=0; i < 50; i++) {
    count++;
    currentX[i]=col;
    col+=25;

    currentY[i]=row;

    if(count==10){
        row+=25;
        col=10;
        count=0;
    }
}
```

This works by looping through each circle position. This in effect draws 10 circles with the Y value of 10. After it has looped through 10 times count will = 10. It will then add 25 to the 'row' value and draw another 10 circles with the Y value of 35. Each loop the X position is also moved across 25 points. It will keep doing this until 50 circles have been given values.

The following two lines of code are used to start the mouse listener "listening" on the applet:

```
addMouseListener(this);
addMouseMotionListener(this);
```

'MouseMotionListener' is used for picking up the motion of the mouse. Things such as the X,Y cords and if the mouse is on the applet or not.

"MouseListener' is used for detecting mouse clicks.

The last section in the 'init()' function is just simply to give all the bullets a position off of the screen so they are hidden and ready to be fired.

```
for(int i=0; i < BNUM; i++){
    bX[i]=-10;
    bY[i]=-10;
    ebX[i]=0;
    ebY[i]=height+10;
}
```

The next function is 'run()'. So many changes have been made to this function that you can basically delete it and I will go through it.

```
while(true){ // Starts the game loop
    start = System.currentTimeMillis(); // Sets the current time
```

```
if(playing==0){ // Are we playing or is the game over?
```

Next section we will move the aliens left and right. It will first move them to the right by adding 1 to step until step is greater than 15. When this occurs it will then set step to 0 and change the direction to 0 which means move them to the left. After it moves 15 positions it will also move down one row.

```
step++;
for(int i=0; i < MAX; i++){
    if(step > 15) {
        if(direction==1){
            direction=0;
        } else {
            direction=1;
        }
        step=0;
        for(int j=0; j < MAX; j++)
            currentY[j]+=speed;
    }
    if(direction==1)
        currentX[i]+=speed;
    else
        currentX[i]-=speed;
}
```

This next for loop is used to tell if the user has fired a bullet. If they have and there is a free bullet (set so only 10 bullets can be fired at once) then to set it to the current ship position. Also if the bullets are visible on the screen then move them up.

```
for(int i=0; i < BNUM; i++){
    if(bY[i] <= 0) {
        bX[i]=mbx;
        bY[i]=mby;
        mbx=-10;
        mby=-10;
    }
    bY[i]-=speed;
}
```

Also related to the bullets is this for loop that detects any collision between the player's bullets and the aliens. This section works by looping through each alien and then each bullet. If the distance between the two is less than 20 then a collision has occurred. The bullet and the alien will then be hidden.

```
for(int i=0; i < MAX; i++){
```

```

    for(int j=0; j < BNUM; j++) {
        if(!(bY[j]!=0)){
            dist = (int)(Math.sqrt(Math.pow((currentX[i]+10)-bX[j],2) +
Math.pow((currentY[i]+10)-bY[j],2)));
            if(dist <= 20){
                bY[j]=-50;
                currentY[i]=-500;
            }
        }
    }
}

```

The next section is used for shooting the alien bullets. It works much the same as the previous shooting section. However this one will randomly pick a alien to shoot from.

```

for(int k=0; k < MAX; k++){
    randomShoot=(int)(Math.random()*MAX);
    if(currentY[randomShoot] >= 0){
        for(int i=0; i < BNUM; i++){
            if(ebY[i] >= height) {
                ebX[i]=currentX[randomShoot];
                ebY[i]=currentY[randomShoot];
                break;
            }
        }
    }
}

```

This is the collision detection section between the alien bullets and the player's ship. Again it is similar to the previous section.

```

for(int j=0; j < BNUM; j++) {
    if(!(ebY[j]>=height)){
        dist = (int)(Math.sqrt(Math.pow((shipX+10)-ebX[j],2) + Math.pow((shipY+10)-
ebY[j],2)));
        if(dist <= 20){
            ebY[j]=height+10;
            health-=10;
        }
    }
}

```

We now need to move all the alien bullets down the screen. I may not have mentioned this before but if you notice that the bullets position 'ebY[]' is moved to the current position plus 'speed'. Everything that is required to move except for the ship is moved by the value 'speed'. I did this so you can change the speed of the game if you wish.

```

for(int i=0; i < BNUM; i++){
    if(ebY[i] < height) {
        ebY[i]+=speed;
    }
}

```

This is simple enough. If the player has no health left then set 'playing' to 2, which means it's "Game Over".

```

if(health <=0)
    playing=2;

```

This is the last section for the game loop. This will detect if all of the aliens have been destroyed, or if the aliens have invaded. If all aliens have been destroyed then set 'playing' to 3 which means the player wins.

```

int count=0;
for(int j=0; j < MAX; j++){
    if(currentY[j]<0)
        count++;

    if(currentY[j]>=340)
        playing=2;
}

if(count==MAX)
    playing=3;

} else {}

repaint(); // Redraw the screen

```

As explained this section calculates the frame rate and how long to sleep for. Please refer to the first tutorial for an explanation.

```

tick_end_time = System.currentTimeMillis();
tick_duration = tick_end_time - start;
sleep_duration = MAX_MS_PER_FRAME - tick_duration;

if (sleep_duration < MIN_SLEEP_TIME)
{
    sleep_duration = MIN_SLEEP_TIME;
}
fps = 1000 / (sleep_duration + tick_duration);

```

```

        try{
            Thread.sleep(sleep_duration);
        } catch(InterruptedExceotion e) {}
    }
}

```

That's the end of our 'run()' function. The next section for us to look at is the drawing section. Its all a lot to take in but don't worry we are on the home stretch, only one more section after this.

As I mentioned these functions are for drawing to the screen. I will go through most of the code again as I don't want to miss anything out:

```

public void paint(Graphics g)
{
    update(g);
}

public void update(Graphics g)
{
    Graphics2D g2 = (Graphics2D)g;

    // Set the background color.
    bufferdImgSurface.setBackground(Color.black);

    // Clear the applet.
    bufferdImgSurface.clearRect(0, 0, width, height);

    bufferdImgSurface.setColor(Color.green);
    //(X pos, Y pos, Width, Height)
    for(int i=0; i < MAX; i++)
        bufferdImgSurface.fillOval(currentX[i], currentY[i], 20,20);

    // Draw the read ship (a square)
    bufferdImgSurface.setColor(Color.red);
    bufferdImgSurface.fillRect(shipX, shipY, 20, 20);
}

```

This is a for loop that will draw all the bullets on the screen. I made it easy by having 10 bullets for the aliens and the player but if you change one of these numbers please be aware that it will cause problems in one place such as this. You would need to split the bullet drawing section into 2 for loops.

```

for(int j=0; j < BNUM; j++){
    bufferdImgSurface.setColor(Color.yellow);
    bufferdImgSurface.fillOval(bX[j], bY[j], 5,5);
}

```

```

        bufferdImgSurface.setColor(Color.blue);
        bufferdImgSurface.fillOval(ebX[j], ebY[j], 5,10);
    }
    // Draw a bottom line to our window
    bufferdImgSurface.setColor(Color.red);
    bufferdImgSurface.drawString("_____
_____ ",0,375);

```

These if statements display the game status, such as if the player loses it will display "****Game Over****".

```

if(playing==1)
    bufferdImgSurface.drawString("PAUSED", width/2-10, 390);
else if(playing==2)
    bufferdImgSurface.drawString("****Game Over****", width/2-10, 390);
else if(playing==3)
    bufferdImgSurface.drawString("****You Win!****", width/2-10, 390);

```

A simple way of displaying a health bar is to loop while the loop value is less than the value in health. On every loop draw a '|'. Set the X position to the current i value multiplied by 2 to give some spacing:

```

    for(int i=0; i < health; i++)
        bufferdImgSurface.drawString(" |", (2*i), 390);

    // Draw the buffered image to the screen.
    g2.drawImage(bufferdImg, 0, 0, this);
}

```

That's the last of our graphics section and now we are onto our last section!
This section is used for detecting mouse clicks and detecting the mouse position.

Move the ship to the current mouse X position:

```

public void mouseMoved(MouseEvent e) { shipX=e.getX()-5; }

public void mouseDragged(MouseEvent e) { shipX=e.getX()-5; }

```

The user clicked a button so start firing a bullet from the current mouse X position and the current ship Y position:

```

public void mouseClicked(MouseEvent e) {
    mbx=e.getX();
    mby=shipY;
}

public void mousePressed(MouseEvent e) {

```

```
mbx=e.getX();  
mby=shipY;  
}
```

The mouse has entered the applet area so set 'playing' to 0 (start the game)

```
public void mouseEntered(MouseEvent e) { playing=0; }
```

The mouse has exited the applet area so set 'playing' to 1 (pause the game)

```
public void mouseExited(MouseEvent e) { playing=1; }
```

We don't use this function but you still must include it in your code or you will have errors:

```
public void mouseReleased(MouseEvent e) { }
```

You should now be able to compile the code and play the game. Its basic I know but it should help with understanding of applets, threads and event handling.

I hope you like my tutorial. If you find any errors please email me, also if you have any comments please email me.

Also check out my website for more tutorials and code samples www.jcroucher.com

This code and tutorial is copyright 2002 John Croucher.