



# **What is static?**

of Amazing Programmers

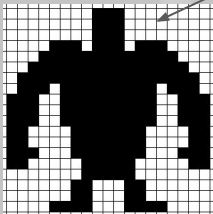
(no static)

```
public class Hero {  
    int strength;  
}
```

When a member variable or method in a Java class is **not** labeled as "static", that means every object of that class will get its own copy of that variable or method.

```
public static void main(String[] args) {  
    Hero warrior = new Hero();  
    Hero ninja = new Hero();  
    warrior.strength = 300;  
    ninja.strength = 10;  
}
```

warrior

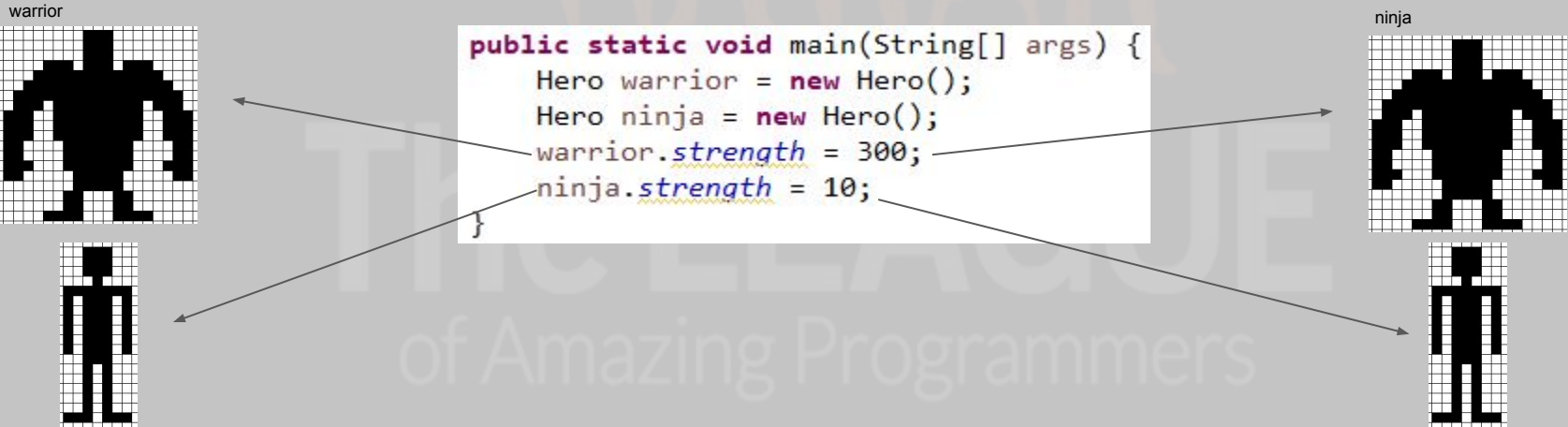


ninja



```
public class Hero {  
    static int strength;  
}
```

If a member variable or method is labeled as "static", only one is created in memory. That means every object of that class will share the same variable or method. If it changes for one object, it changes for all of the objects.



Since static member variables and methods are shared between all of the objects created by that class, we don't need an object to access them. They can be accessed just by using the class name.

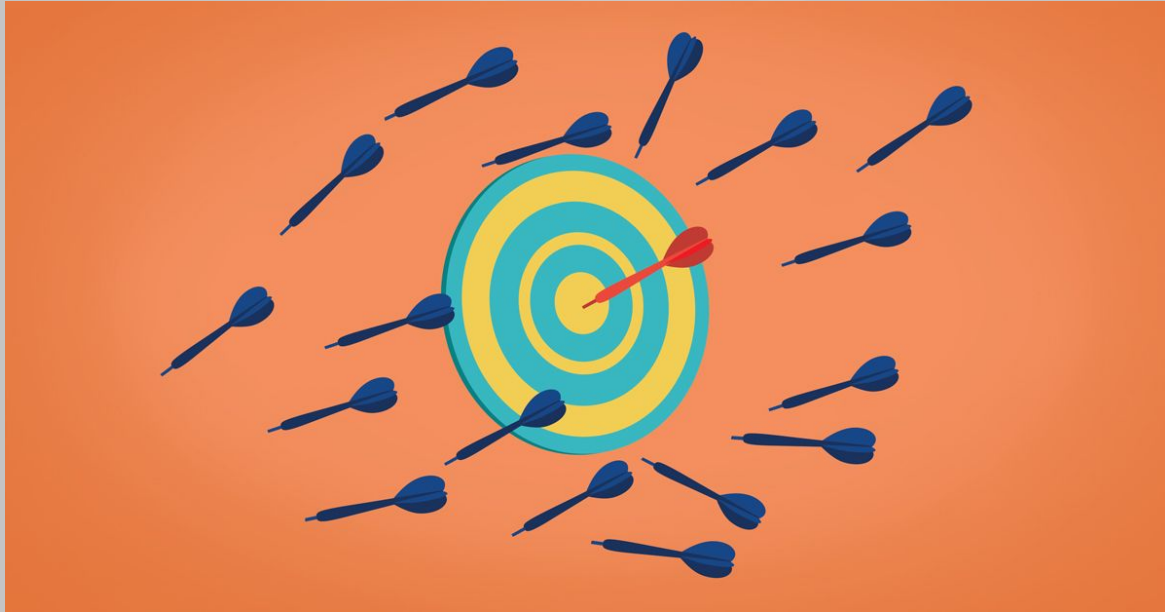
```
public static void main(String[] args) {  
    Hero warrior = new Hero();  
    Hero ninja = new Hero();  
    Hero.strength = 300;  
    Hero.strength = 10;  
}
```

changes strength to 300 for  
both Hero objects

changes strength to 10 for  
both Hero objects

THE LEAGUE  
of Amazing Programmers

Let's practice!



of Amazing Programmers

Take a few minutes to study *World.java* file in your project.  
Notice that every member variable and method is static.

```
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.HashMap;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class World {
    private static JFrame window;
    private static JPanel panel;
    private static BufferedImage worldImage;
    private static HashMap<Entity, Integer> entities = new HashMap<Entity, Integer>();

    public static void show() {
        if(window != null) {
            window.dispose();
        }
        window = new JFrame();
        panel = new JPanel() {
            public void paintComponent(Graphics g) {
                g.drawImage(worldImage, 0, 0, 700, 700, null);

                for(Entity e : entities.keySet()) {
                    int xy = entities.get(e);
                    int x = xy >> 16;
                    int y = xy & 0xffff;
                    g.drawImage(e.image, x, y, 50, 50, null);
                }
            }
        };

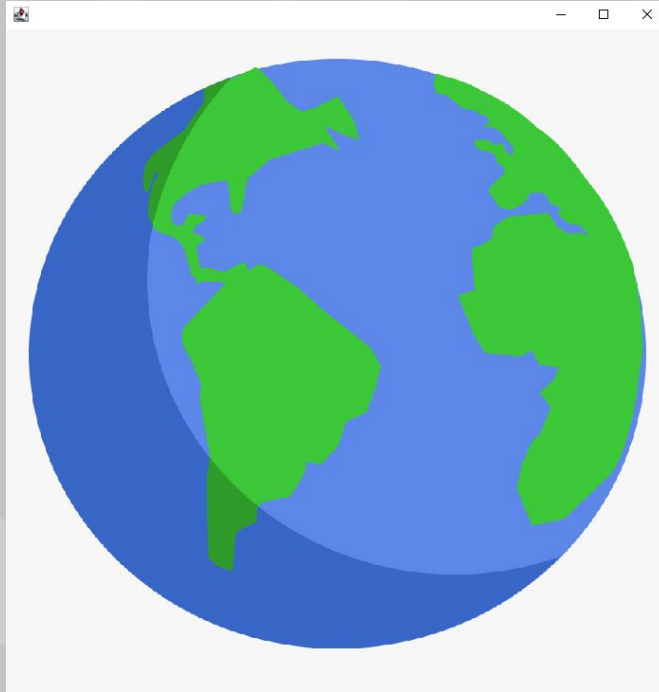
        try {
            worldImage = ImageIO.read(new World().getClass().getResourceAsStream("earth.png"));
        } catch (IOException e) {
            e.printStackTrace();
        }

        panel.setPreferredSize(new Dimension(700, 700));
        window.add(panel);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.pack();
        window.setVisible(!window.isVisible());
    }

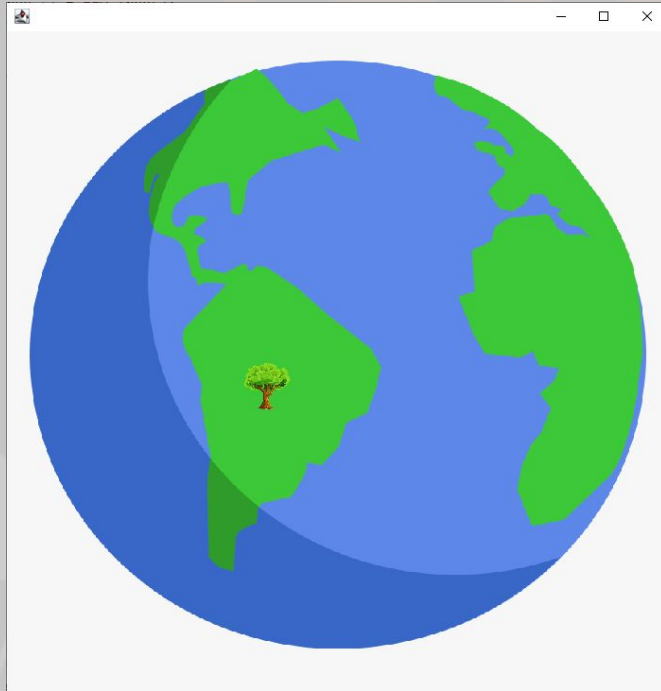
    public static void addEntity(Entity e, int x, int y) {
        int xy = (x << 16) | (y & 0xffff);
        entities.put(e, xy);
    }

    public static void reset() {
        entities.clear();
    }
}
```

In the Runner class, add code to make the world show.

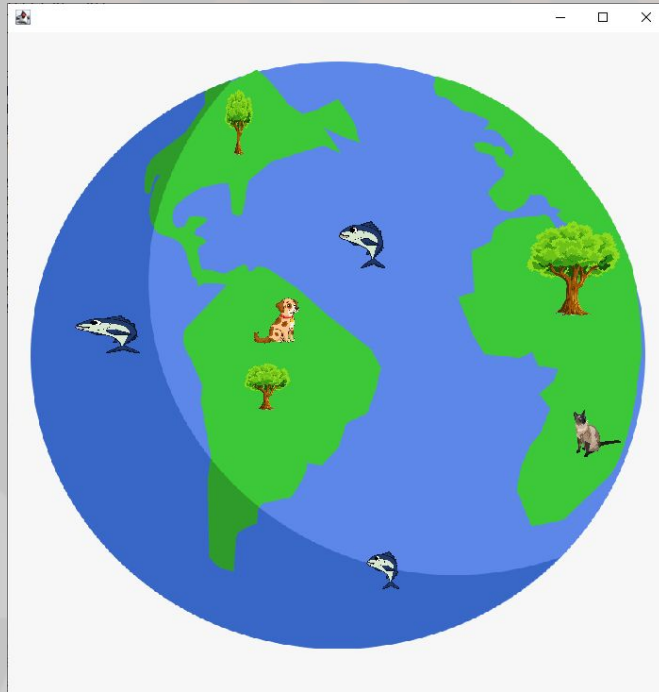


Next, create a tree object and add it to the world. Make sure the code to show the world goes last.



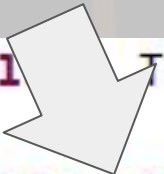


Create more entities and populate your world with them.



Notice that in each Entity subclass, the *image* member variable is **static**. We don't need to load a copy of the same image when rendering the same entity. They can all share the same image.

So by marking it as **static**, each entity of the same type will share an image. This saves memory!



```
public class Tree extends Entity {  
    private static BufferedImage image;  
  
    public Tree(int w, int h) {
```

of Amazing Programmers

What are some other things that static member variables and methods can be useful for?



