Statics and Mains

- .java and .class
- Static initialization
- Instance initialization
- Mains and args

See Also

http://csis.pace.edu/~bergin/KarelJava2ed/ch2/javamain.html

https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html

.java and .class

- Every class has a compiled ".class" file
- Usually a class is defined in a separate ".java" file
- A ".java" file can only have one "public" class
- A public class must be defined in "name" .java
- When you compile a ".java" file the compiler recompiles all ".java" files that have changed

Static Blocks

```
public class Point {

    static {
        System.out.println("I am here");
    }

}

public static void main(String [] args) {

    System.out.println("TINKER");

    Point p = new Point();
    Point q = new Point();

    System.out.println("Done");

}
```

```
@ Javadoc  Probler
<terminated> Tinker (3)
TINKER
I am here
Done
```

- Static blocks execute when class loads
- Executed only once

Static Initialization

```
public class Point {

    static int count;

    static {
        count = 5;
    }

}
```

```
public class Point {

    static int count = 5;

}
```

- Static variables are like globals. The ONE copy is kept with the class.
- Don't have to have an instance to use the static (global) members.
- Compiler generates a "static" block for you.

Instance Initialization

```
public class Point {

    int a = 5;

}
```

```
class Point {

    int a;

    {
        a = 5;
    }

    public Point() {
        a = 5;
    }

}
```

- You can initialize member variables "inline" or in a constructor.
- The compiler will move the initialization to the constructor(s) for you.
- These two produce the same code.

Mains

```java
public class Point {

   int a;

   public Point() {
       a = 5;
   }

   public static void main(String[] args) {
       System.out. println("Point:"+Arrays.toString(args));
   }

}
```

```java
public class Tinker {

    public static void main(String [] args) {

        System.out.println("Tinker:"+Arrays.toString(args));

    }

}
```

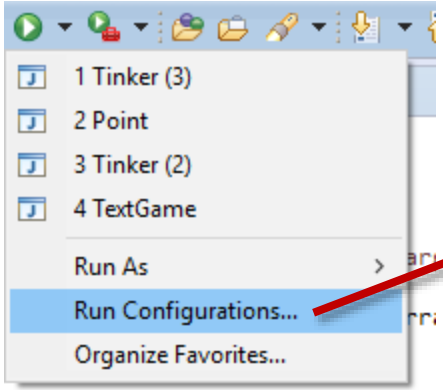
- Every class can have a “main”.
- You pick which “main” to run.

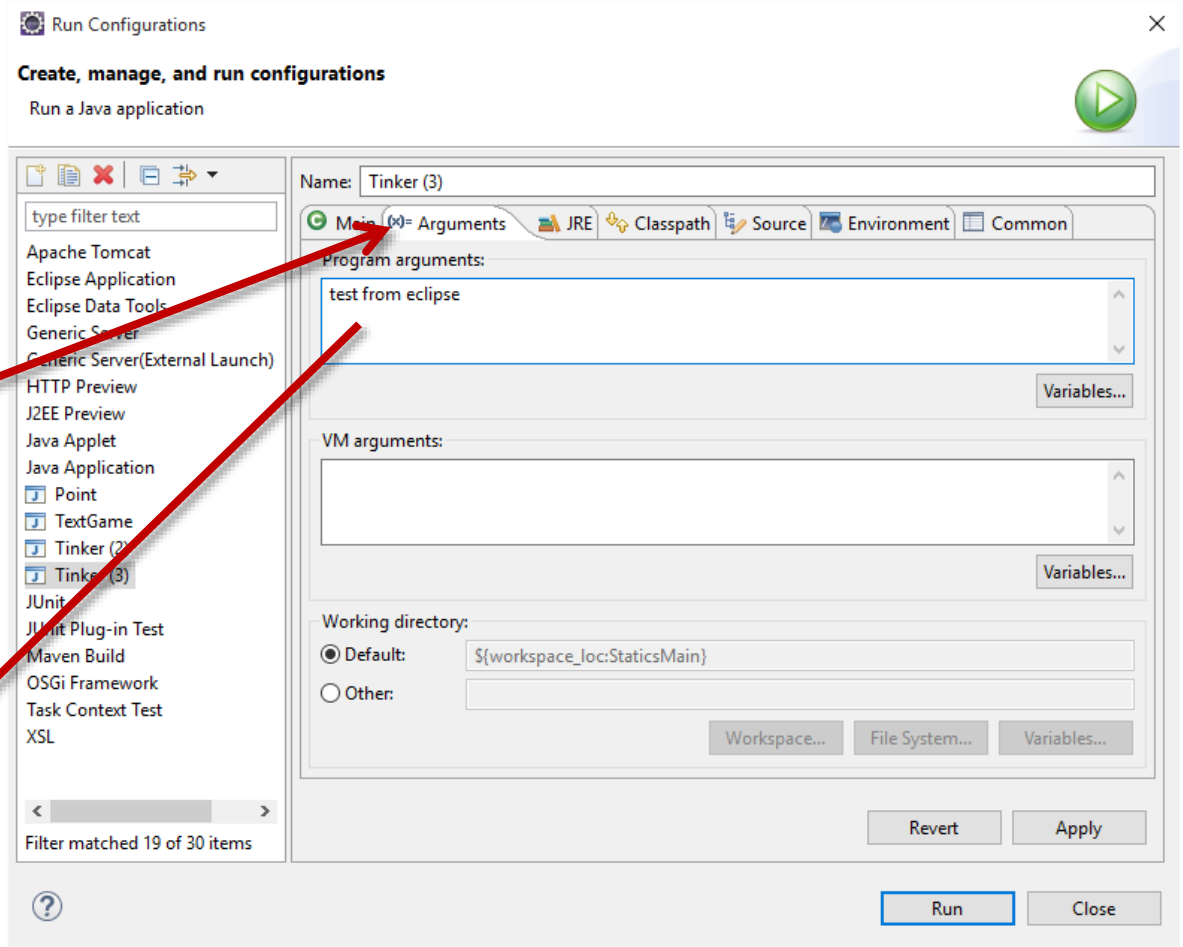
Command Prompt

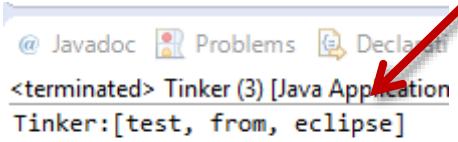
```
I:\test>java Point how now brown cow
Point:[how, now, brown, cow]

I:\test>java Tinker hello world
Tinker:[hello, world]
```

In Eclipse


1 Tinker (3)
2 Point
3 Tinker (2)
4 TextGame
Run As
Run Configurations...
Organize Favorites...


Run Configurations
Create, manage, and run configurations
Run a Java application
type filter text
Apache Tomcat
Eclipse Application
Eclipse Data Tools
Generic Server
Generic Server(External Launch)
HTTP Preview
J2EE Preview
Java Applet
Java Application
Point
TextGame
Tinker (2)
Tinker (3)
JUnit
JUnit Plug-in Test
Maven Build
OSGi Framework
Task Context Test
XSL
Filter matched 19 of 30 items
Name: Tinker (3)
Main
Arguments
JRE
Classpath
Source
Environment
Common
Program arguments:
test from eclipse
Variables...
VM arguments:
Variables...
Working directory:
Default:
${workspace_loc:StaticsMain}
Other:
Workspace...
File System...
Variables...
Revert
Apply
Run
Close


Javadoc
Problems
<terminated> Tinker (3) [Java Application
Tinker:[test, from, eclipse]

Mains Calling Mains

```java
public class Tinker {

    public static void main(String [] args) {

        System.out.println("Hello");

        Point.main(null);

        String [] p = {"From", "Tinker"};

        Point.main(p);

        System.out.println("Done");

    }

}
```

@ Javadoc | Problems | Declaration | Console

```
<terminated> Tinker (3) [Java Application] C:\Program Files\Java
Hello
Point:null
Point:[From, Tinker]
Done
```

Tinkering

- Create three classes – each with a main.
- Run the mains one by one from Eclipse.
- Print the command line arguments passed to each.
- Use Eclipse to pass in command line arguments.
- Make one main call the other two.