Virtual Threads — Permanent in 21

- Delivered in JDK 19 last September
- Response has been enthusiastic and results are promising
- API is finalized
- Pinning — decided not to wait
- Structured Concurrency and Scoped Values — Preview
- JEP 444: Virtual Threads

The legendary “sufficiently good GC” is here

- JEP 439: Generational ZGC

Starting Small

Ron Pressler

Starting Small

- Java is the leading language for big, long-lasting, server-side programs because it’s great at scaling *up*
- Java has lost ground in education and in smaller software because it’s not so great at scaling *down*
- Every large project starts out small
- Every expert starts out a beginner
- Incumbents are always disrupted from *below*

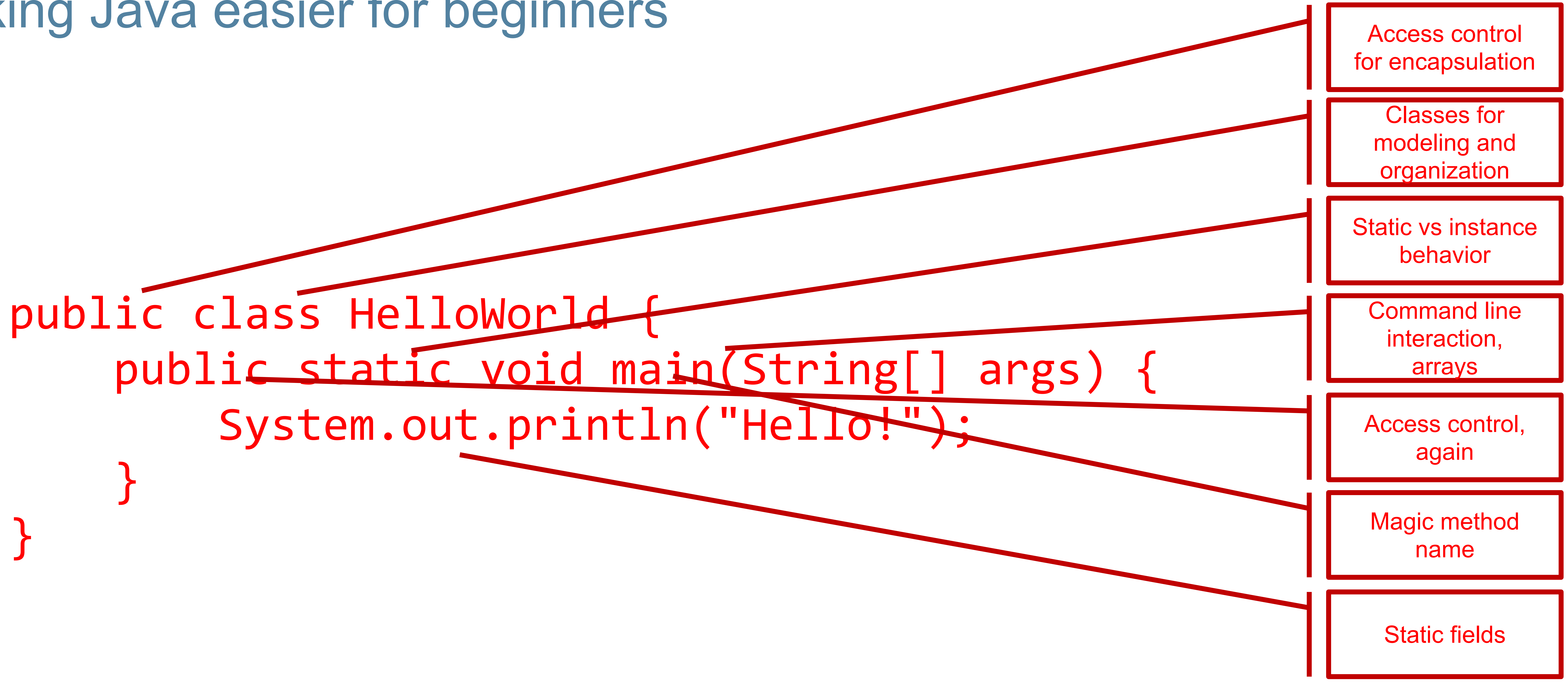
Starting Small

- Reduce effort to learning for beginners, as well as for starting a project for experts
- Do not introduce a separate “beginners’ dialect” of Java
- Do not introduce a special tooling workflow for beginners
- Changes must be a *natural*, consistent evolution of the Java language and tooling
- A series of independent JEPs covering the language, existing tools, and new tools

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello!");
    }
}
```

Paving the on-ramp

Making Java easier for beginners



```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello!");
    }
}
```

Access control for encapsulation
Classes for modeling and organization
Static vs instance behavior
Command line interaction, arrays
Access control, again
Magic method name
Static fields

Programming in the large and in the small

- **Programming in the large** composing encapsulated components with interfaces
- **Programming in the small** component internals
- What's large and small is relative
  - Module — an unnamed one is provided implicitly
  - Package — an unnamed one is provided implicitly
  - Class — an unnamed one will now be provided implicitly
- Access modifiers are a mechanism for programming in the large

Anonymous Main Classes

- Must declare a `main` method outside a class declaration
- Must be in the unnamed package
- Cannot declare a constructor
- Unnamed, so cannot be accessed directly from other classes; it can only be launched
- Body allows the same syntax with the same meaning as an anonymous class today

```java
public static void main(String[] args) {
    System.out.println("Hello!");
}
```

`static`

- `static` is an OOP detail of classes and objects
- It is viral: for a static `main` to call `foo`, `foo` must either be static, or `main` must construct an object.
- We allow an instance `main` method when the class has a non-private no-args constructor.
- This automatic construction has a precedent in lambdas

```
public void main(String[] args) {
    System.out.println("Hello!");
}
```

Program Entry Point

- `public void main(String[] args)` is arbitrary;
  may as well have been `int entry(List<String> args)`

- We will now allow `void main()`

```
void main() {
    System.out.println("Hello!");
}
```

```java
void main() {
    System.out.println(greeting("World!"));
}

String prefix = "Hello, ";

String greeting(String who) {
    return prefix + who;
}
```

Even experts write simple programs; this change increases the signal/noise ratio

Anonymous Main Classes and Enhanced Main Methods

- The two features are orthogonal
  - An ordinary class can use an instance `main`
  - An anonymous class can use a static `main`

Paving the on-ramp

Making Java easier for beginners

```
void main() {
    println("Hello!");
}
```

Starting Small

- Even big projects done by experts start small — tinkering and exploration
- JShell (JEP 222, integrated in JDK 9) — tinkering with statements
- Launch Single-File Source-Code Programs (JEP 330, JDK 11) — tinkering with one file
- Once we have more than one file we configure a build tool

Launch Multi-File Source-Code Programs

- Let programmers choose when they want to set up a build configuration
- We will allow launching *multi-file* source code programs, without a compilation step

```
// - Prog.java
class Prog {
    public static void main(String[] args) { Helper.run(); }
}

// - Helper.java
class Helper {
    static void run() { System.out.println("Hello!"); }
}

// - lib1.jar
// - lib2.jar
```

```
java -cp '*' Prog.java
```

Launch Multi-File Source-Code Programs

- Works when source files span multiple packages
- Works when source files span a single module
- Works even with dynamically-loaded classes (`Class.forName`)
  - A custom class-loader compiles sources on-demand

Starting Small

- What about downloading and using libraries?
- What about choosing and learning a build tool?

How to follow this?

- JEP 445: Flexible Main Methods and Anonymous Main Classes (Preview)
- JEP draft: Launch Multi-File Source-Code Programs