CS 251
Intermediate Programming
Keyword Roundup

Brooke Chenoweth

University of New Mexico

Spring 2025

So many keywords!

- static
- final
- abstract
- public, private, protected
- class, interface, enum
- extends, implements
- this, super
- instanceof
- try, catch, finally, throw, throws

Primitive Types

- boolean
- byte
- char
- double
- float
- int
- long
- short

Literal values: true, false, null

- true, false, and null are not actually keywords[1], they are literals
- Still can't use them as identifiers in your programs

[1] splitting hairs here

var for type inference

- Java 10 introduced a reserved type name[2] for inferring types of local variables.
- Only works when initializing a local variable to something non-null
- Can make more readable code by eliminating redundant type information. Instead of

```
BigUglyLongTypeName a = someMethod();
AnotherUglyType b = new AnotherUglyType(a);
oneMoreMethod(b);
```

We can use

```
var a = someMethod();
var b = new AnotherUglyType(a);
oneMoreMethod(b);
```

[2] not technically a keyword, so code with variables named `var` still OK

void

- Use `void` when declaring/defining methods to specify method has no return type.

```
public void myMethod() {
}
```

- `void` is not a type. (Can't have a void reference, unlike C or C++)

Control Flow

- break
- case
- continue
- default[3]
- do
- else
- for
- if
- return
- switch
- while

[3] Also can allow interface to provide implementation of a method

new

Use new to create new objects.

- `Color col = new Color(5,5,5);`
- `Block[] blocks = new Block[20];`

`static`

If a member is *static*, it belongs to the class, not a particular instance.

- Variables
- Methods
- Nested classes
- Nested interfaces
- Nested enums

`final`

If something is *final*, it cannot be changed.

- A final variable cannot be reassigned.
  - Final member variable is initialized once when object is constructed.
  - Final parameter is not assigned within method.
  - Final local variable will not be reassigned.
  - Local class method referencing local variable or parameter insists on final.
- A final method cannot be overridden.
- A final class cannot be extended.

abstract

- An abstract method is declared without an implementation.
- If a class has abstract methods, it must be declared abstract. (May also declare class abstract without abstract methods.)
- An abstract class cannot be be instantiated.

this

- Access member variables.
- Call one constructor from another.

```
public class Point2D {
  protected double x, y;
  public Point2D(double x, double y) {
    this.x = x;
    this.y = y;
  }
  public Point2D() {
    this(0, 0);
  }
}
```

super

- Call parent constructor from child's constructor.
  - Must be first statement in child constructor.
  - If omitted, default (no argument) parent constructor is called.
- Access parent methods when child overrides.
- Access hidden parent fields. (Tip: Don't hide fields. It's confusing!)

```
public class Point3D extends Point2D {
  protected double z;
  public Point3D(double x, double y, double z) {
    super(x, y);
    this.z = z;
  }
}
```

Access Modifiers

`private`	Only this class
package-private	No modifier. This class and others in same package
`protected`	This class and its subclasses, plus same package
`public`	Accessible to all

package

- Use packages to group related types.

```
package mypackage;
public class MyClass {
}
```

- Companies use reversed internet domain name to begin package names.
  - `com.example.mypackage`
- If no package statement, your type will be in an unnamed package.
- Package structure and directory structure usually must match.

import

- Can use package member with fully qualified name without importing.
  - `mypackage.MyClass`
  - `java.awt.Color`
- Can import specific member

```
import mypackage.MyClass;
```

- Can import entire package with wildcard.

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

`class`, `interface`, `enum`

- A class is a template for a type of object.
- An interface is contract defining public behaviour.
- An enum is a special kind of class that defines a fixed set of constants.
- All can be top-level or nested.

record classes for plain data

As of JDK 16, we have a new keyword `record` to define a simple object type to hold a few fields of immutable data.

```
public record Rectangle(double length, double width) { }
```

is equivalent to

```
public final class Rectangle {
  private final double length;
  private final double width;

  public Rectangle(double length, double width) {
    this.length = length;
    this.width = width;
  }

  double length() { return this.length; }
  double width()  { return this.width; }

  // equals(), hashCode(), toString using fields
}
```

`extends` vs `implements`

- A class `extends` its parent class.
- An interface possibly `extends` another interface.
- A class `implements` an interface.

`instanceof`

- Sometimes, you really need to know if an object is an instance of a particular type. You can use `instanceof` to find out.
- Reconsider your design if you are using lots of `instanceof` expressions. Polymorphism is better!
- Common use of `instanceof` is when overriding `equals`

Pattern matching with `instanceof`

Let's assume with have a `Shape` interface, implemented by `Rectangle`

```
if( shape instanceof Rectangle ) {
  Rectangle r = (Rectangle)shape;
  // do Rectangle stuff here
}
```

The `instanceof` check tells us we can safely cast.

Pattern matching with `instanceof`

Let's assume with have a `Shape` interface, implemented by `Rectangle`

```
if( shape instanceof Rectangle ) {
  Rectangle r = (Rectangle)shape;
  // do Rectangle stuff here
}
```

The `instanceof` check tells us we can safely cast. As of JDK 17, we can simplify this test plus cast like so:

```
if( shape instanceof Rectangle r ) {
  // do Rectangle stuff here
}
```

Exception Handling

- `try` – clause for testing potential exception code.
- `catch` – catching the exceptions, if they happen
- `throws` – used in method headers to indicate method might cause exception
- `throw` – used by a method to "throw" (cause) an exception
- `finally` – code executed after the try-catch clauses, regardless of whether exception happened or not.

Other Keywords – not used in this course

- `transient` – used to exclude field from serialization
- `synchronized` – used in multithreaded programs
- `volatile` – used in multithreaded programs
- `const, goto` – not actually used, but are reserved words
- `assert` – used to test assumptions (disabled by default)

assert

- If you want to programmatically test your assumptions, you can add `assert` statements to your program.
- `assert booleanExpr;`
- By default, does nothing, but if you run with the -ea flag to enable assertions, will throw an `AssertionError` if `booleanExpr` is false.
- You would only use this in testing, not for actual program control. (Since they can be disabled.)