Java Review I

Idaho State University | Computer Science

Dr. Isaac Griffith

CS 2263
Department of Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will:

- Understanding the different types of files used in Java development
- Understand how the JVM works with these different files
- Understand the basic tools provided by the JDK
- Understand and be able to define the different types, including:
  - Primitives
  - Classes (including Abstract and Sealed Classes)
    - Enums
    - Records
  - Interfaces

What's in it for Me?

Before we start this lecture, I would like you to:

1. Take out a sheet of paper or open a text editor
2. Write the following:
   - One thing you feel you know well from the list of outcomes
   - Two things you want to know more about from the list of outcomes.
3. Pause the lecture and complete the list.
4. When you are finished, resume the lecture.

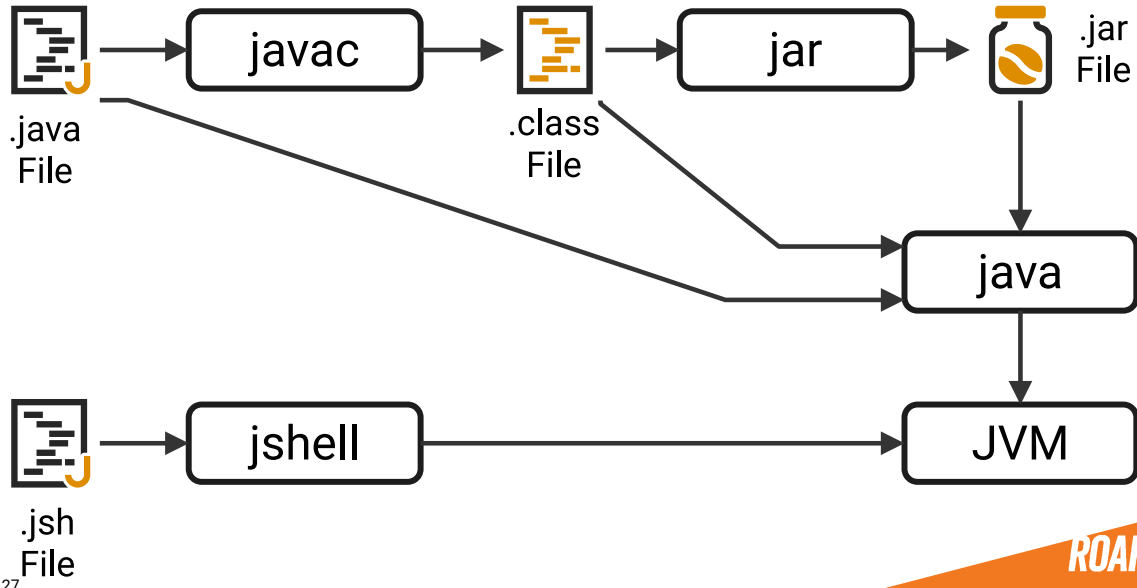
Files, JVM, and JDK

CS 2263

ROAR

Idaho State University
Computer Science

JVM and Java Files



ROAR

JDK Tools

- java - executes a class, java source file, or jar file on the JVM
- javac - compiles java source files to class files
- jar - compresses a collection of classes into a single java archive, JAR, file
- jshell - provides a REPL for executing scripts written in Java

Java OO Basics

CS 2263

OO Basics

- There are five basics things needed for a language to support Object-Oriented Programming:
  - Inheritance
  - Encapsulation
  - Abstraction
  - Polymorphism
  - Delegation

Inheritance

- We can inherit attributes and methods from one class to another.
- There are two categories:
  - **subclass**(child) - the class that inherits from another class
  - **superclass**(parent) - the class being inherited from
- There are several ways in which this can be done
  - a class or record may inherit from exactly one class using the `extends` keyword
  - an enum cannot extend another class (it already inherits from Enumeration)
  - an interface may inherit from another interface using the `extends` keyword
  - a class may realize an unlimited number of interfaces using the `implements` keyword

Inheritance

- Sealing a class restricts a class to be inherited from by a specific subset of classes
  - done using the `sealed` keyword and the list is provided using the `permits` keyword
- Inheritance passes down all `public` and `protected` methods
- Non-abstract classes must implement all abstract methods not implemented by parent classes
- Final classes provide the ability to prevent inheritance

Inheritance Examples

Basic

```
public class Vehicle {}

public class Car extends Vehicle {}
```

Interface Inheritance

```
public interface Drivable {}

public interface Stoppable extends Drivable {}
```

Interface Realization

```
public interface Drivable{}

public class Car extends Vehicle implements Drivable {}
```

Sealed

```
public abstract sealed class Vehicle implements Drivable permits Car {}

public class Car extends Vehicle {}
```

Encapsulation

Definition

To ensure that ”sensitive” data is hidden from users

- Facilitated by:
  - declaring class variables/attributes as `private`
  - providing public **get** and **set** methods to access and update the values of the private variables

Example

```
public class Person {
  private String name;

  public String getName() { return name; }

  public void setName(String name) { this.name = name; }
}
```

Abstraction

- Data abstraction is the process of hiding certain details and showing only essential information to the user.
- Can be achieved using either (which we will discuss later):
  - Abstract Classes
  - Interfaces

Examples

```
public abstract class Test {
  // contents
}

public interface Other {
  // contents
}
```

Polymorphism

- **Polymorphism:** having many forms
- Polymorphism is the ability of a message to be displayed in more than one form
- Occurs when we have classes related to each other via inheritance
- Facilitated by inheriting behavior, which can be called across an inheritance hierarchy
- Polymorphic methods are those which can be executed on a number of related objects
  - By calling on the object
  - By passing objects in as parameters

Examples:

```
public abstract class Shape {
  public abstract double area();
}

public class Square extends Shape {
  public double area() { /* ... */ }
}

public class Circle extends Shape {
  public double area() { /* ... */ }
}

public void calcArea(Shape s) {
  // Do something
}
```

Coupling

- **Coupling:** When two classes have a relationship. There are several types of couplings:
  - *Association*: When a class has a field whose type is the other class
  - *Usage*: When a class has a method that either
    - Returns a value with the type of the other class
    - Has a parameter with the type of the other class
    - Declares a local variable with the type of the other class
  - *Generalization*: When a class extends the other class
  - *Realization*: When a class realizes the other interface

Composition and Delegation

Composition: The inclusion of one or more objects within another object

- forming a part-whole type relationship.
- Typically formed by using a collection of the contained objects.
- **Goal:** The containing object utilizes the contained objects to perform its functions via delegation.

Example:

```
class Vehicle {
  Engine engine;
  Wheel[][] wheels = new Wheel[2][2];
  //...
}
```

Delegation: Where an object's method provides its functionality by calling another method on another object.

Example:

```
Other other;

public String delegate() {
  return other.target();
}
```

Defining Types

CS 2263

ROAR

Types

Primitives - basic data types which can hold only a single value

- Numeric primitives (bits):
  - `byte` (8)
  - `short` (16)
  - `int` (32)
  - `long` (64)
  - `float` (32)
  - `double` (64)
- Other:
  - `boolean` - `true` or `false`
  - `char` - a 16-bit unicode character

Example:

```
byte ex1 = 100;
boolean ex2 = true;
char ex3 = 'c';
double ex4 = 1.0;
```

Objects - instances of class data types which are more complex

Example:

```
String s = "string";
```

Type Definitions

- In java we can define new types in the following four ways:
  - Using Classes
  - Using Interfaces
  - Using Enums
  - Using Records

Classes

- Provide the basic structural building block in Java
- Provide the ability to combine data and behavior together into a single unit
- Give us the ability to define what objects will look like

Syntax

```
[access] [modifiers] class Identifier [extends Type] [implements ImplementsList] [permits PermitsList]
```

- Access:
  - `public`: can be used by any class, must have same name as the containing file
  - default: can only be used by classes in the same package
  - `private`: can only be used by the class in which this type is defined in
- Modifiers:
  - `final`: notes that this class is not abstract and cannot be extended by inheritance (thus cannot be sealed either)
  - `sealed`: notes that the extension of this class is restricted to the types in the PermitsList
  - `abstract`: notes that this class cannot be instantiated, and requires being extended

Example

```
public sealed class TestA extends Test
    permits TestB, TestC {
    // Members
}
```

Abstract Classes

- A restricted class that cannot be used to create objects
- Typically contains one or more **abstract methods**
- Often used to achieve security

Example:

```
abstract class Animal {
  public abstract void animalSound();
  public void sleep() {
    System.out.println("Zzz");
  }
}
```

Interfaces

- Another form of abstraction in Java
- All methods (excluding default methods) are assumed to be `public` and `abstract`, thus have no method body.
- All fields are assumed to be constants, thus are `public static final`
- When implementing an interface
  - You either need to implement all methods (excluding default methods)
  - Or, declare the implementing class `abstract`

Syntax:

```
[access] interface Identifier [extends OtherInterface] {
  // members
}
```

Example:

```
public interface Animal {
  public void animalSound();
  public void run();
}
```

Enums

- A special type of class with a predefined set of instances
  - these instances are called enum literals and are unchangeable

Syntax:

```
[access] enum Identifier [implements IntefaceList] {
  LITERALS;

  // Other members
}
```

Example:

```
public enum Level {
  LOW(1),
  MEDIUM(2),
  HIGH(3);

  private int value;

  public Level(int value) {
    this.value = value
  }
}
```

Records

- A restricted form of a class
  - Cannot extend any class
  - Cannot declare instance fields (other than private final fields)
  - Cannot be abstract and are implicitly `final`
  - All components of a record are implicitly `final`
- Ideal for data objects where in the data is not meant to be altered
  - Class is final
  - All fields are final
  - Only have simple getter methods and constructors

Syntax:

```
[access] record Identifier(Type field,
         Type field, ...) {}
```

Example

```
final class Rectangle implements Shape {
    final double length;
    final double width;

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

    double length() { return length; }
    double width() { return width; }
}
```

This class can be written as a record as follows:

```
record Rectangle(float length, float width) { }
```

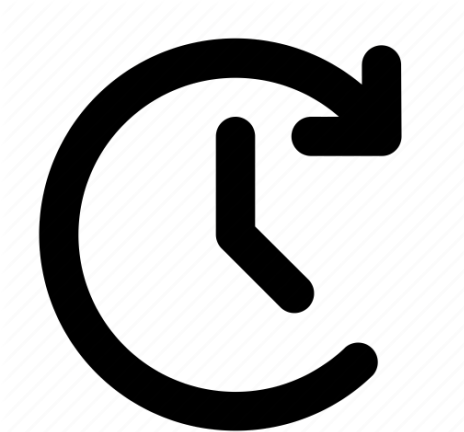
Closing

In closing, let's return to your paper from the beginning of the lecture:

- Ask yourself the following questions:
  1. For the item you felt you knew well:
     - Did you learning anything new?
     - Did you know it as well as you thought?
  2. For the items you wanted to know more about:
     - Did you learn anything new about those concepts?
     - Did you find that you knew more than you thought?

For Next Time

- Review the Appendix and Chapter 2.1 - 2.2
- Review the Lecture
- Read Chapter 2.3 - 2.6
- Come to Class
- Continue working on Homework 01



Are there any questions?