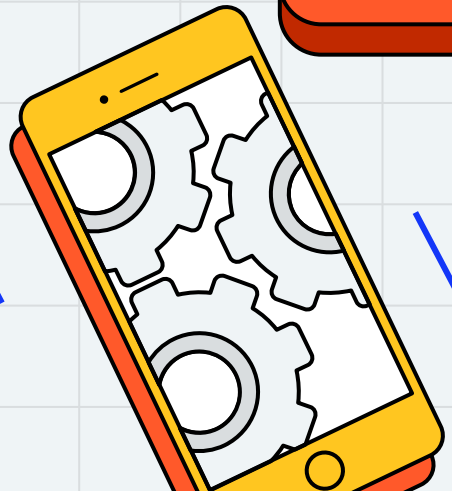
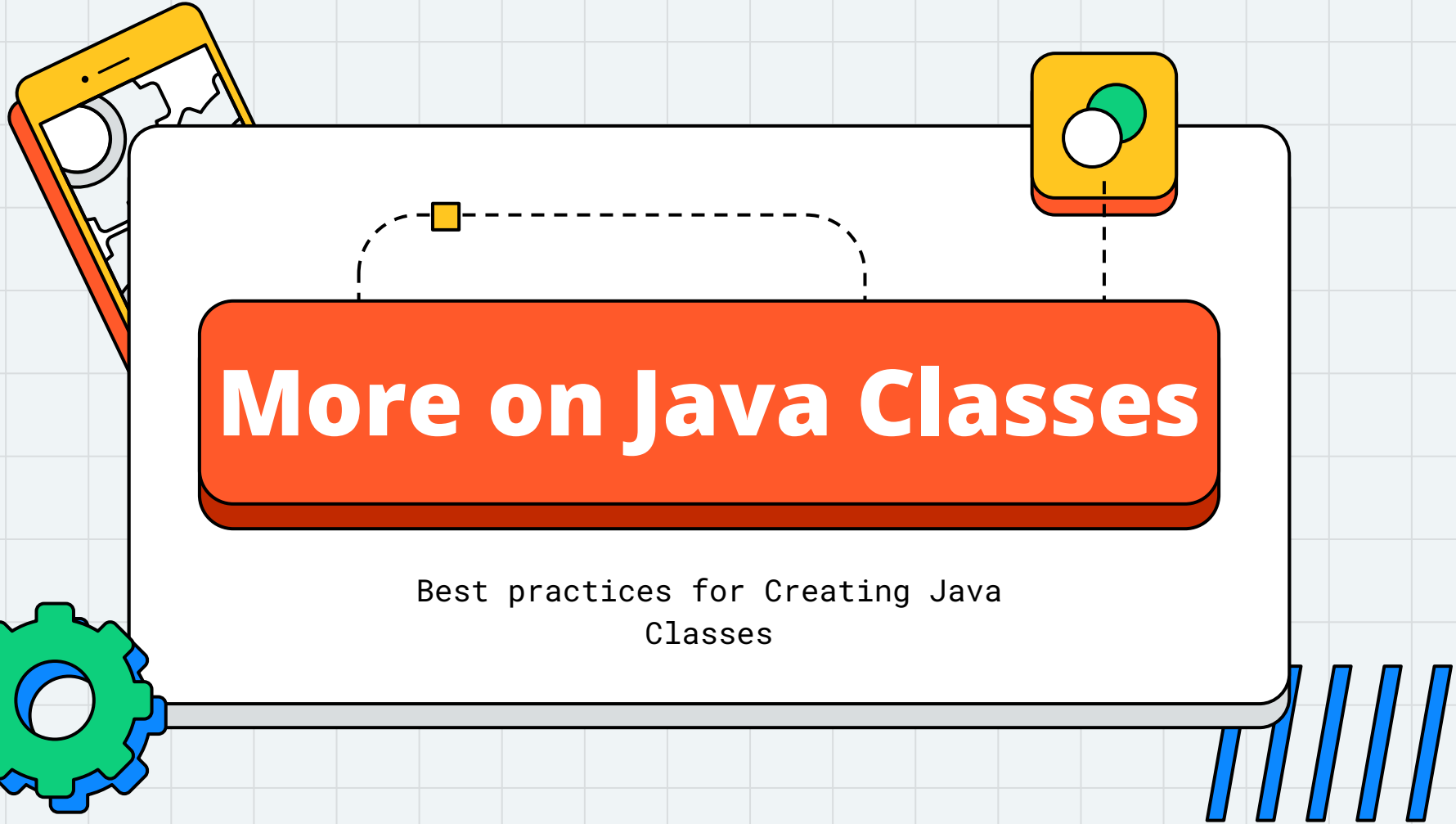
Application Programming



Hend Alkittawi


More on Java Classes
Best practices for Creating Java Classes

BEST PRACTICES FOR CREATING JAVA CLASSES

STYLE

- identifier naming
- indentation

METHODS

- getters and setters
- equals()
- toString()

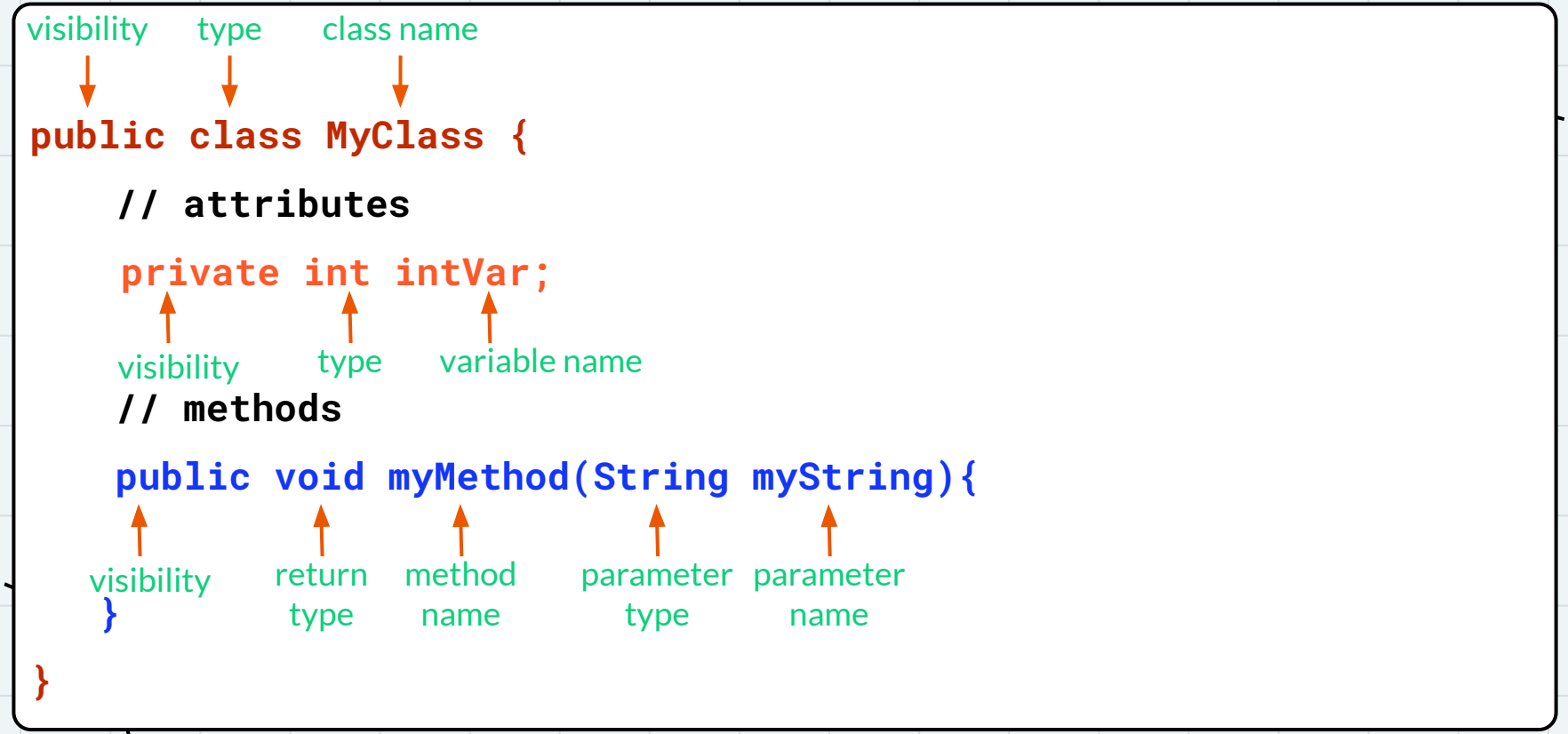
CONSTRUCTORS

- Ways to instantiate objects

JAVADOC

- “special” Java documentation

STYLE


visibility
type
class name
public class MyClass {
// attributes
private int intVar;
visibility
type
variable name
// methods
public void myMethod(String myString){
visibility
return type
method name
parameter type
parameter name
}
}

METHODS

- The syntax (rules) for declaring a **method**

  visibility returnType methodName(parameterList)

  - visibility determines access
    - public (all can access) or private (just this class can access)
  - returnType is the type of data this method returns
    - if nothing is returned, use the keyword void
  - methodName starts with a lowercase word
    - it uses uppercase for the first letter of each additional word (this is called “camel case”)
  - parameterList is any data we need to pass to the method
    - The ordering must be followed exactly

METHODS

- Getter and setter methods are used to get and set the **value of the attributes.**
  - Getters retrieve the value only
  - Setters update the value only

```
public class Account {

    private double balance;
    private String name;

    public void setBalance( double balance ){
        this.balance = balance;
    }
    public double getBalance(){
        return this.balance;
    }
    // add a getter and a setter for name
}
```

- to use the class and its methods

```
public static void main(String[] args){
    Account myAccount = new Account();
    myAccount.setBalance(1000.0);
    double balance = myAccount.getBalance();
}
```

METHODS

- **toString()** method returns a String representation of objects.

  An example of a toString() method for the Account class:

```
public class Account {
  private double balance;
  private String name;
  public void setBalance( double balance ){
      this.balance = balance;
  }
  public double getBalance() {
      return this.balance;
  }
  // … getter and a setter for name
  public String toString() {
    return "Account info: name: " + name
                + " with balance: " + balance;
  }
}
```

- to use the class and its methods

```
public static void main(String[] args){
    Account myAccount = new Account();
    myAccount.setName("Sam Adams");
    myAccount.setBalance(1000.0);
    System.out.println( myAccount );
}
```

METHODS

- **equals()** method provides a way for users to compare instances of your object to other instances. This also gives you control over what is relevant to differentiate your objects.

```
public class Account {

  private double balance;
  private String name;

  public void setBalance( double balance ) {
    this.balance = balance; }

  public double getBalance() {
    return this.balance; }

  public String toString() {
    return "Account info: name: " + name
               + " with balance: " + balance; }

  public boolean equals( Account account2 ) {
    return this.getName().equals( account2.getName());}
}
```

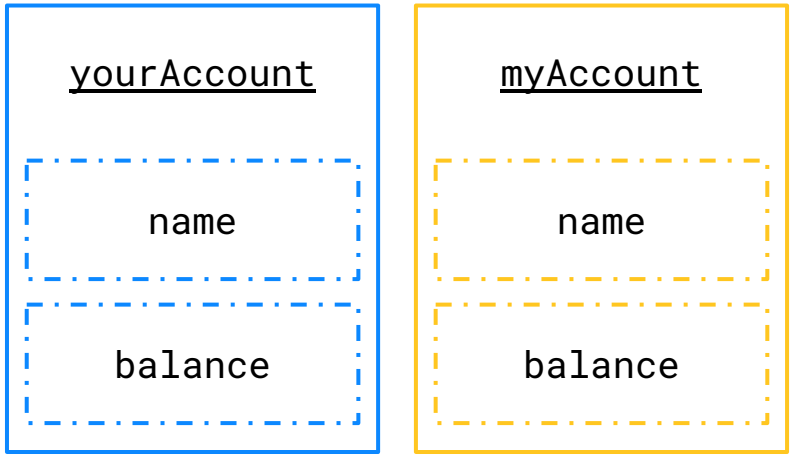
- to use the class and its methods

```
public static void main(String[] args){
  Account myAccount = new Account();
  Account yourAccount = new Account();
  myAccount.setName("Mia");
  myAccount.setBalace(10);
  yourAccount.setName("Ken");
  yourAccount.setBalance(100);
  boolean check =   myAccount.equals(yourAccount);
}
```

IMPORTANT

- **== is used for primitive types only**
  - **2 == 5**
- **Objects define an object-method called equals()**
  - **objA.equals(objB);**
- Core: Drawing Memory Models with Objects - Memory Models, Scope, and Starting the Project | Coursera

```java
public class Account {

  private double balance;
  private String name;

  public void setBalance( double balance ) {
    this.balance = balance; }

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

  public double getBalance() {
    return this.balance; }

  public String getName() {
    return this.name; }

  public boolean equals( Account account2 ) {
    return this.getName().equals( account2.getName());
  }

  public String toString() {
    return "Account info: name: " + name
               + " with balance: " + balance; }
}
```




```java
public static void main(String[] args){
  Account myAccount = new Account();
  Account yourAccount = new Account();
  myAccount.setName("Mia");
  myAccount.setBalace(10);
  yourAccount.setName("Ken");
  yourAccount.setBalance(100);
  boolean check =   myAccount.equals(yourAccount);
}
```

CONSTRUCTORS

- Java requires a constructor call for every object that is created.
- The keyword new creates a new object of a specified class by calling a constructor.
- A constructor is similar to a method but is called implicitly by the new operator to initialize an object's instance variables when the object is created.
- In any class that does not explicitly declare a constructor, the compiler provides a default constructor (which always has no parameters).
- When a class has only the default constructor, the class's instance variables are initialized to their default value.

METHODS

- You can provide your own constructor to specify custom initialization for objects of your class.
- A constructor must have the same name as the class.
- If you declare a constructor for a class, the compiler will not create a default constructor that class.

```
public class Account {

  private double balance;
  private String name;

  public Account( String name, double balance ){
    this.name = name;
    this.balance = balance;
  }
  /* … rest of class …  */
}
```

- to use the class and its methods

```
public static void main(String[] args){
  Account myAccount = new Account("Sam", 1000);
  Account yourAccount = new Account("Jane", 2000);
}
```

JAVA GARBAGE COLLECTION

- More than one variable may refer to the same data.
- Java will clear out old data that no variables are referencing
  - This is known as garbage collection
- **Garbage collection** is the process through which Java will eventually clear out old data that no variables are referencing.

JAVADOC

- Javadoc comments are delimited between /** and */.
- Javadoc comments enable you to **embed program documentation** directly in your programs.
- Javadoc utility program reads Javadoc comments and uses them to prepare program documentation in HTML web-page format.

```
/**
* This is a Javadoc comment!
*/
```

JAVADOC

- Javadoc comments annotations:
  - @author: designates the author of the code, belongs in the class comment
  - @param: designates the parameter to a method, belongs in all method comments which require parameters
  - @return: designates the returned value of a method, belongs in method comments which return values

JAVADOC

Javadoc

```java
/**
* The Account class represents a bank account.
* @author CS3443
*/
public class Account {
    /* attributes here .. a multi-line comment starts with /* .. be careful! */
```

Multiline Comment

```java
    /**
    * sets the account balance
    * @param balance, the account balance (double)
    */
     public void setBalance(double balance){
         this.balance = balance;
     }
```

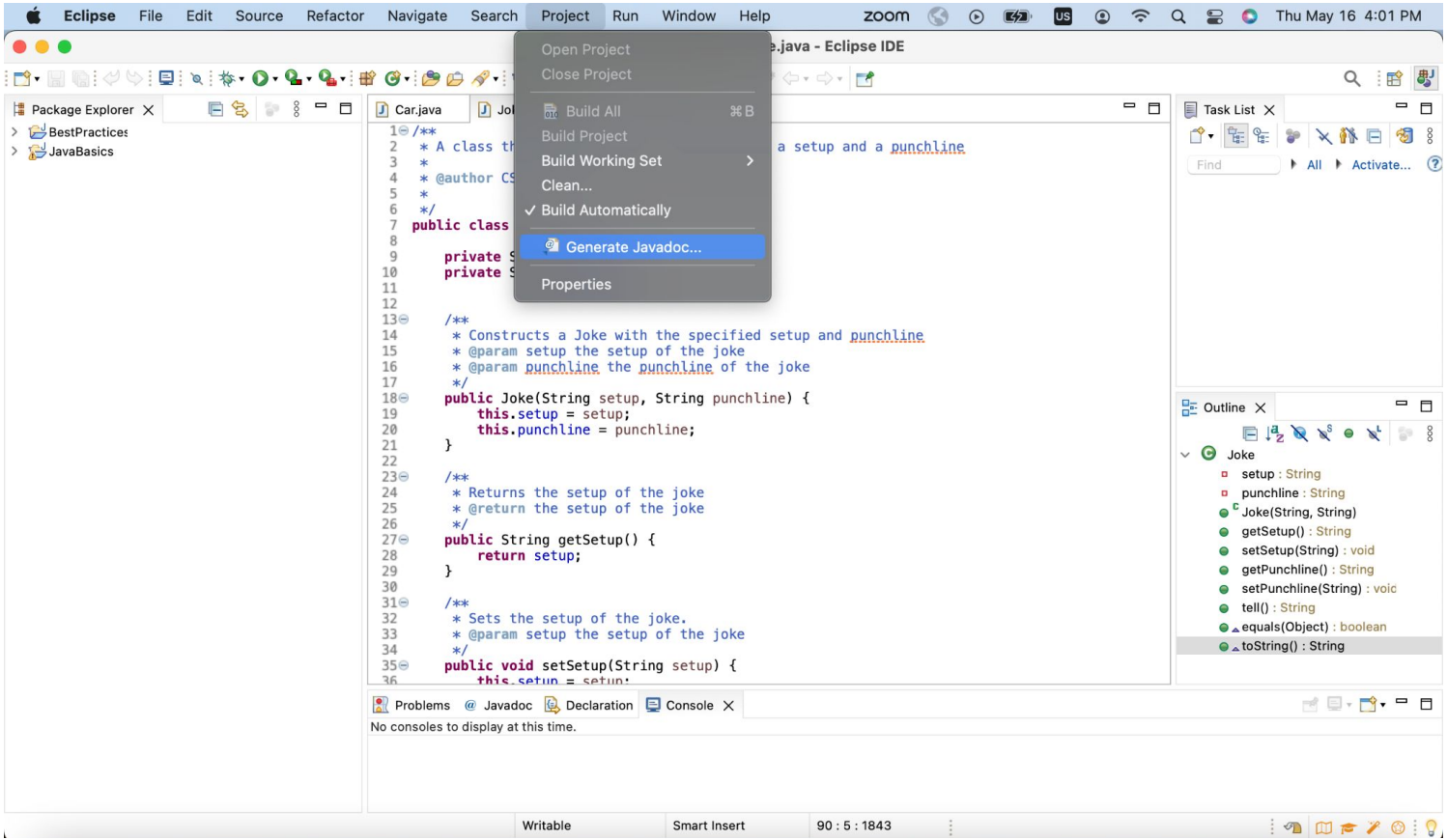
Javadoc

```java
     /**
     * returns the balance for the account
     * @return double, the account balance
     */
     public double getBalance(){
         return balance;
     }
}
```

JAVADOC

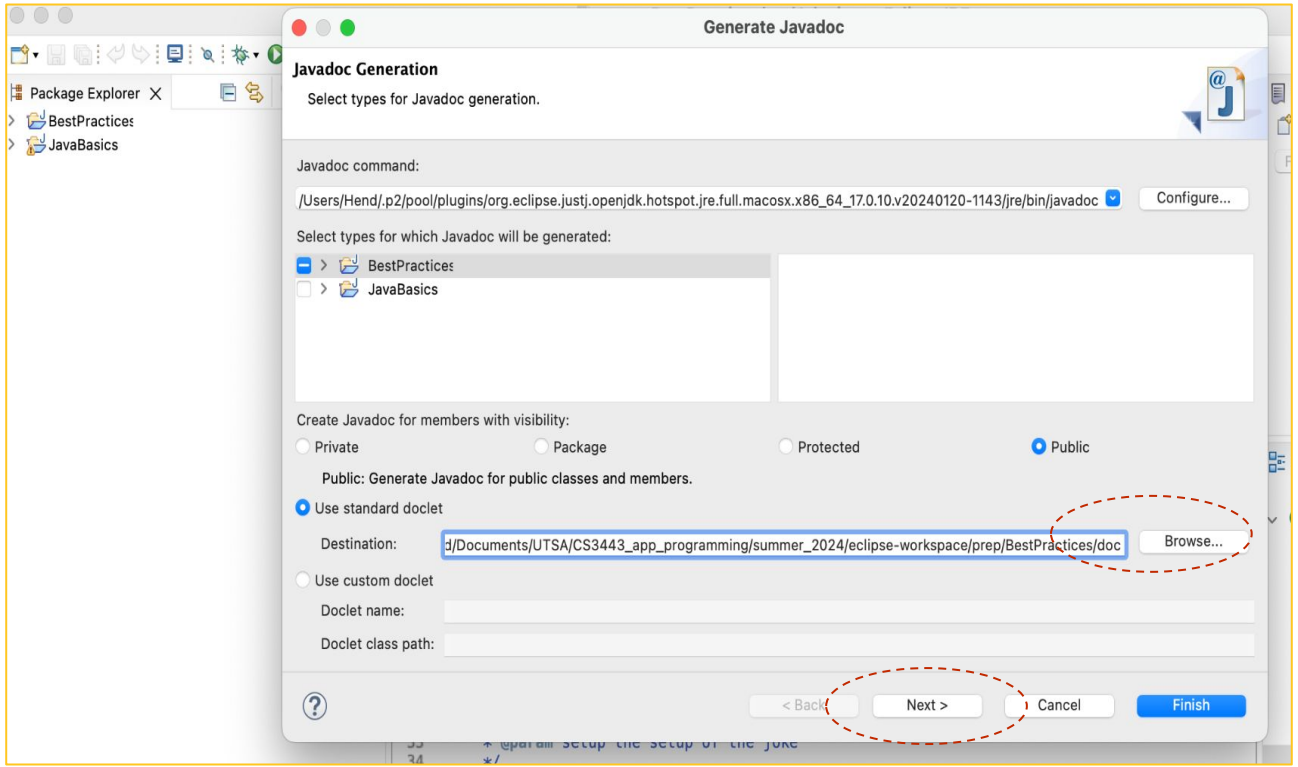
- To generate Javadoc in Eclipse
  1. Project > Generate Javadoc
  2. Destination: workspace/your_project/doc
  3. Next
  4. Select all “referenced archives and projects”
  5. Finish > Yes To All
  6. Open index.html

Project > Generate Javadoc



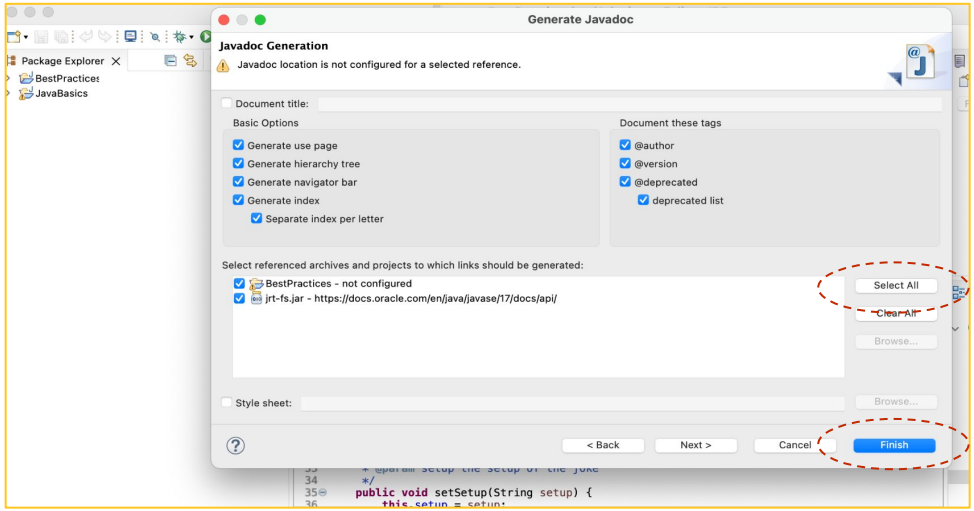
Destination:workspace/your_project/doc

> Next

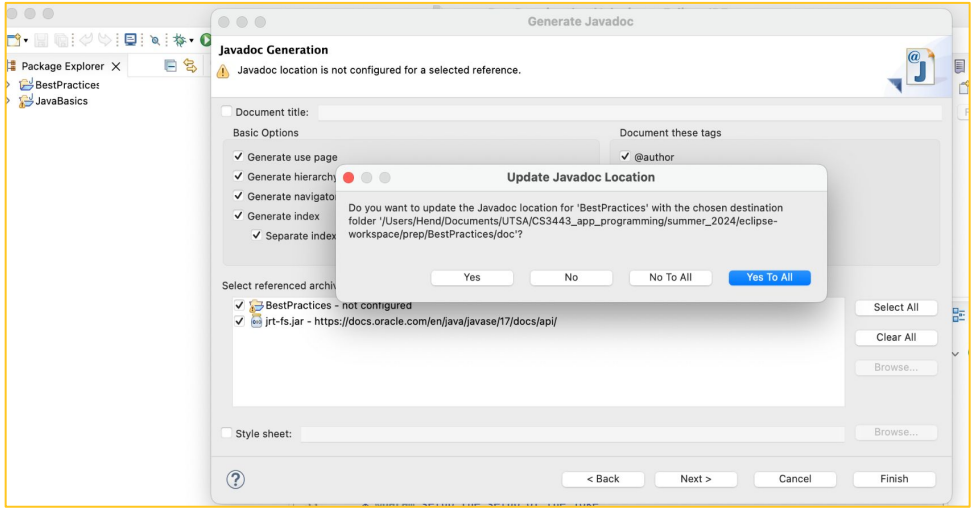


Select all

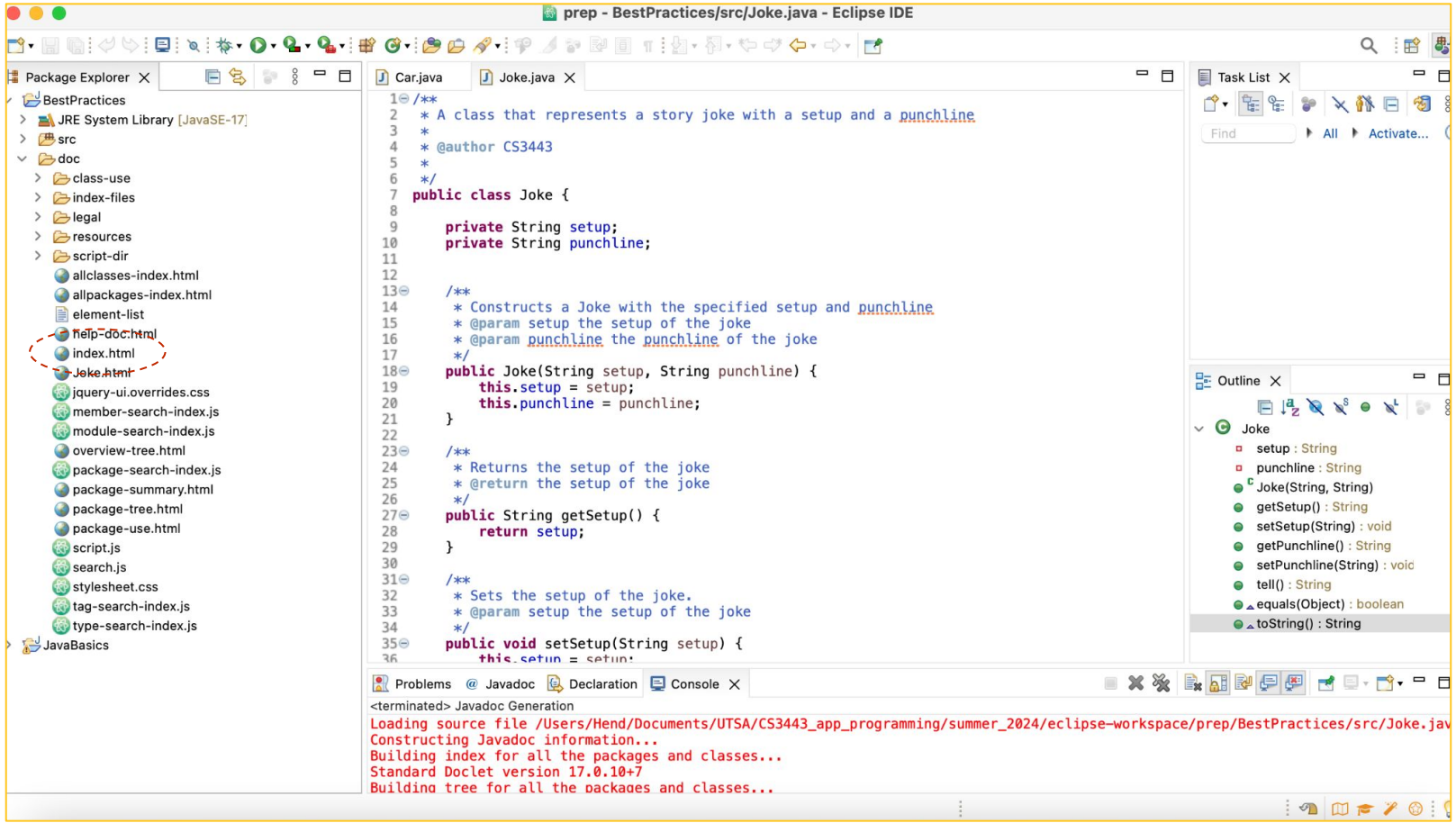
> Finish



Yes To All

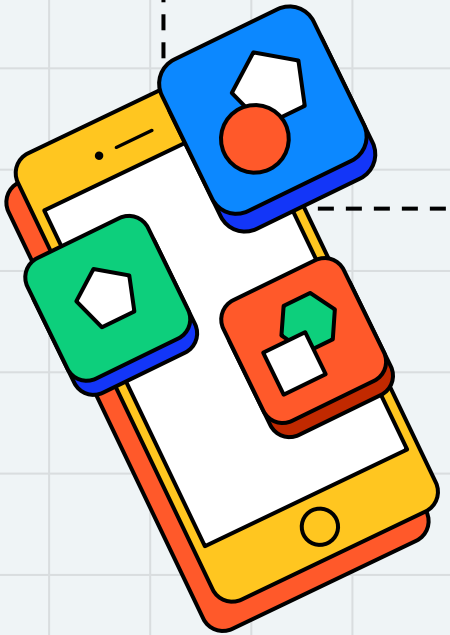


Open index.html



CODE DEMO

- Create a Java class following the best practices
- Generate Javadocs



THANK YOU!

DO YOU HAVE ANY QUESTIONS?

hend.alkittawi@utsa.edu

By Appointment

Online