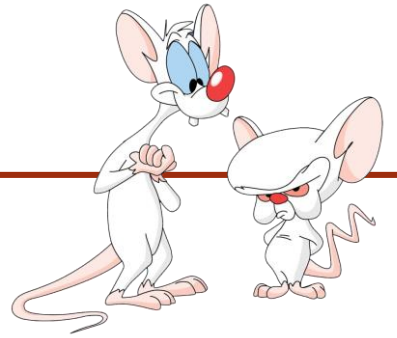
COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Week 3-2: OOD: Packages, Fields, and Modifiers

Giulia Alberini, Fall 2020

WHAT ARE WE GOING TO DO IN THIS VIDEO?



OOD1

- Packages
- Fields
- Modifiers

WARNING!

- This is a terminology-heavy lecture!

- Do not hesitate to stop the video and rewind.

PACKAGES

PACKAGES

- A package is a group of classes
  - Each class is referred to as a *package member*
- A class is a group of methods
- A method is an ordered group of commands

DEFINITION

- To define a package we write at the top of our class file the following statement

```
package packageName;
```


- For example:


```
package nba.annoyingTeams;

public class MiamiHeat {
    ⋮
}
```

This creates a class `MiamiHeat` **inside the package** `nba.annoyingTeams`

FILE AND FOLDERS NAMES

There are two main rules related to files' and folders' names in Java:

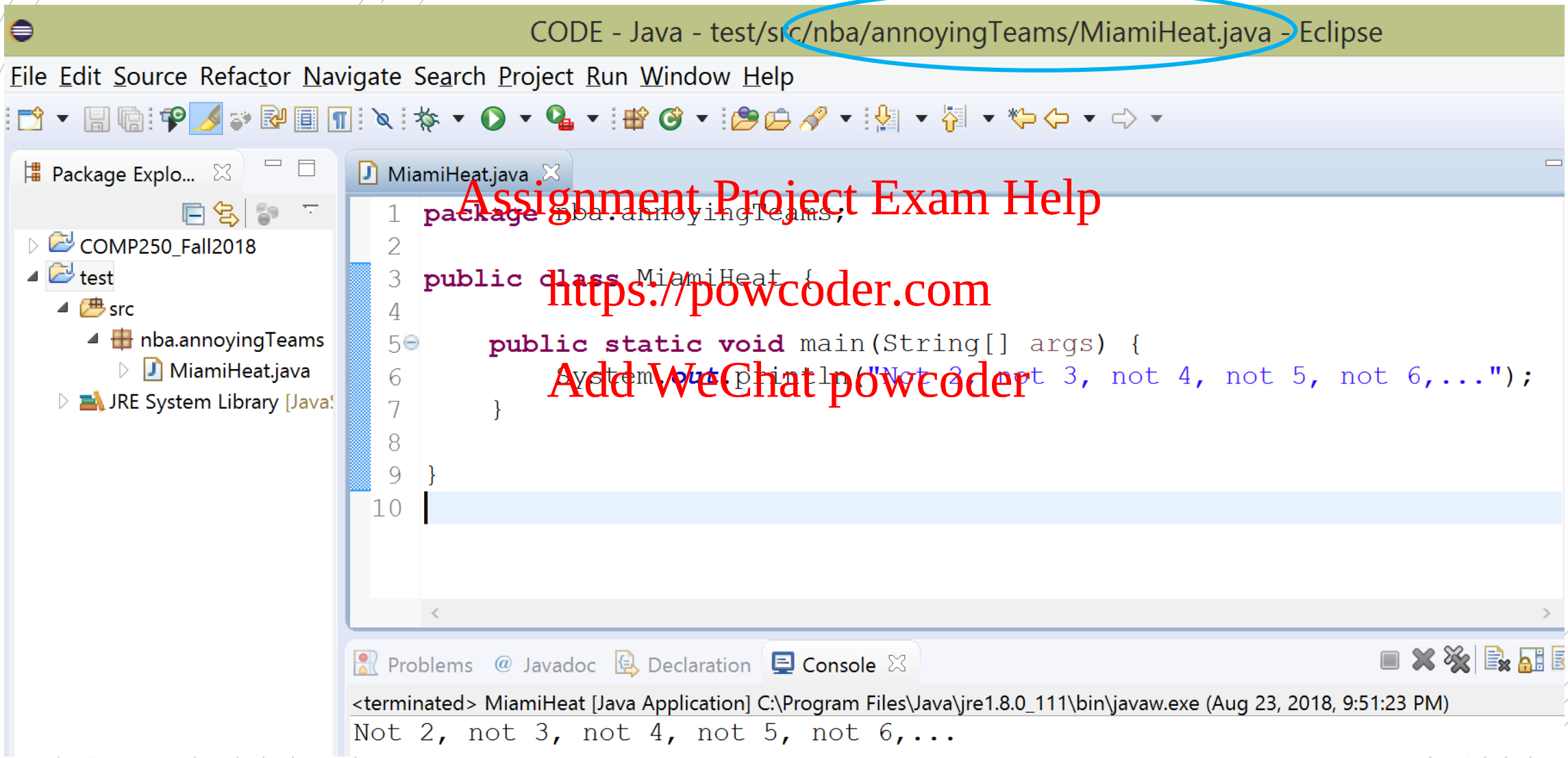
1. The name of the *class* must match the name of the file (with .java added) (e.g. *MiamiHeat.java*)

2. The folder path must match exactly the package name – except that each period is actually a "slash" (i.e. a subfolder)

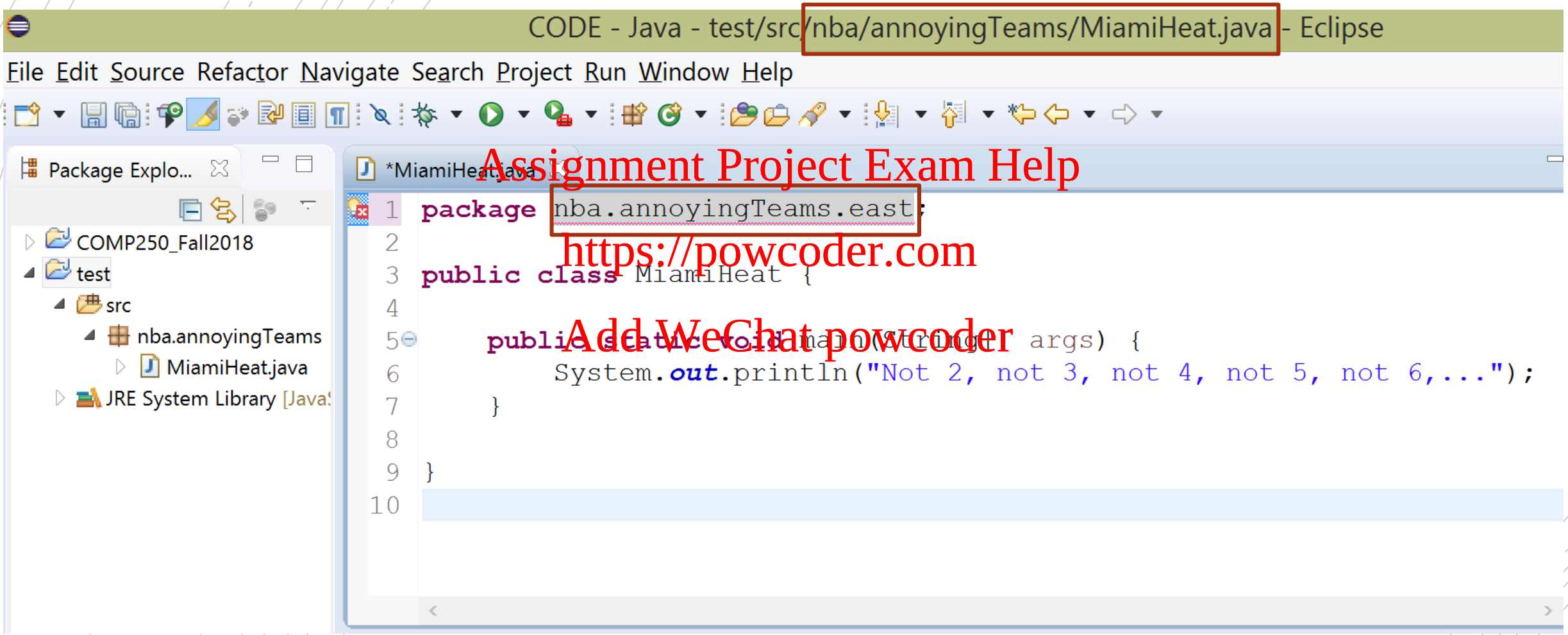
In the example before, a folder *nba* must contain a folder *annoyingTeams* which contains the file *MiamiHeat.java*

https://youtu.be/e9BqUBYaHlM?t=1m53s

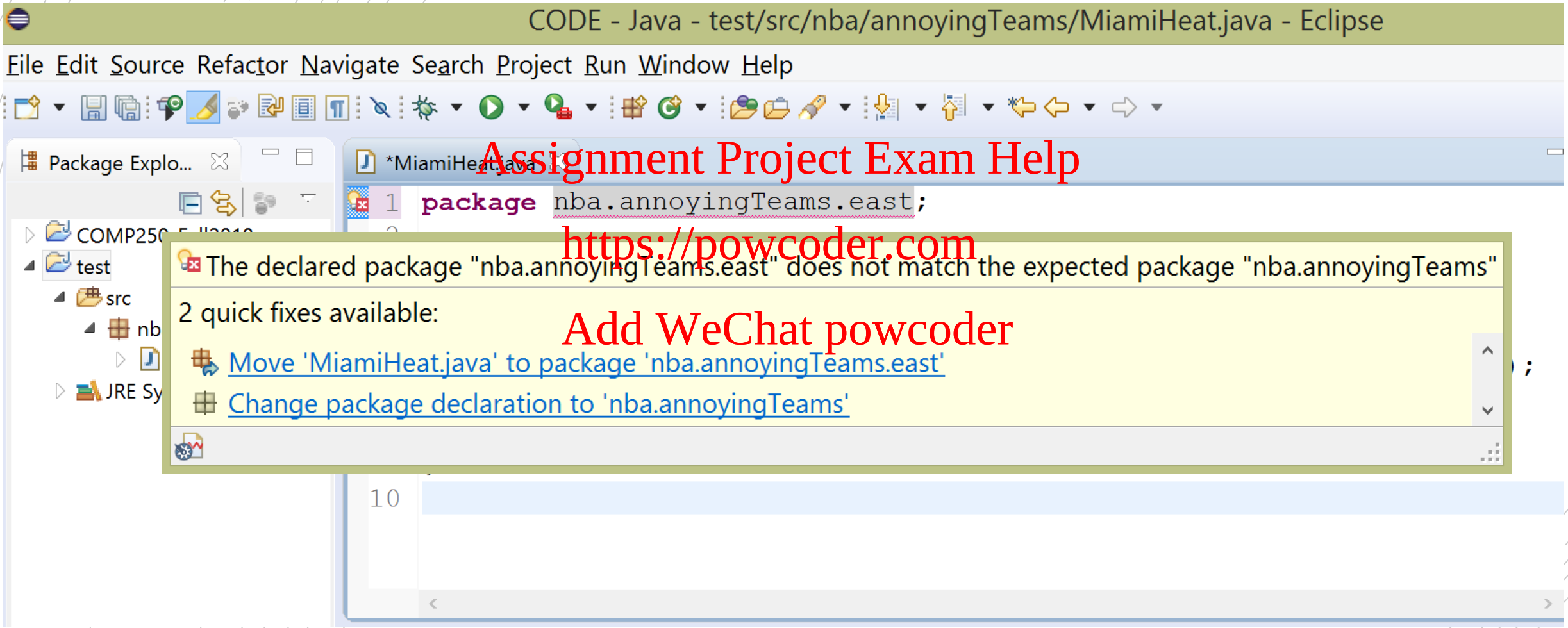
EXAMPLES



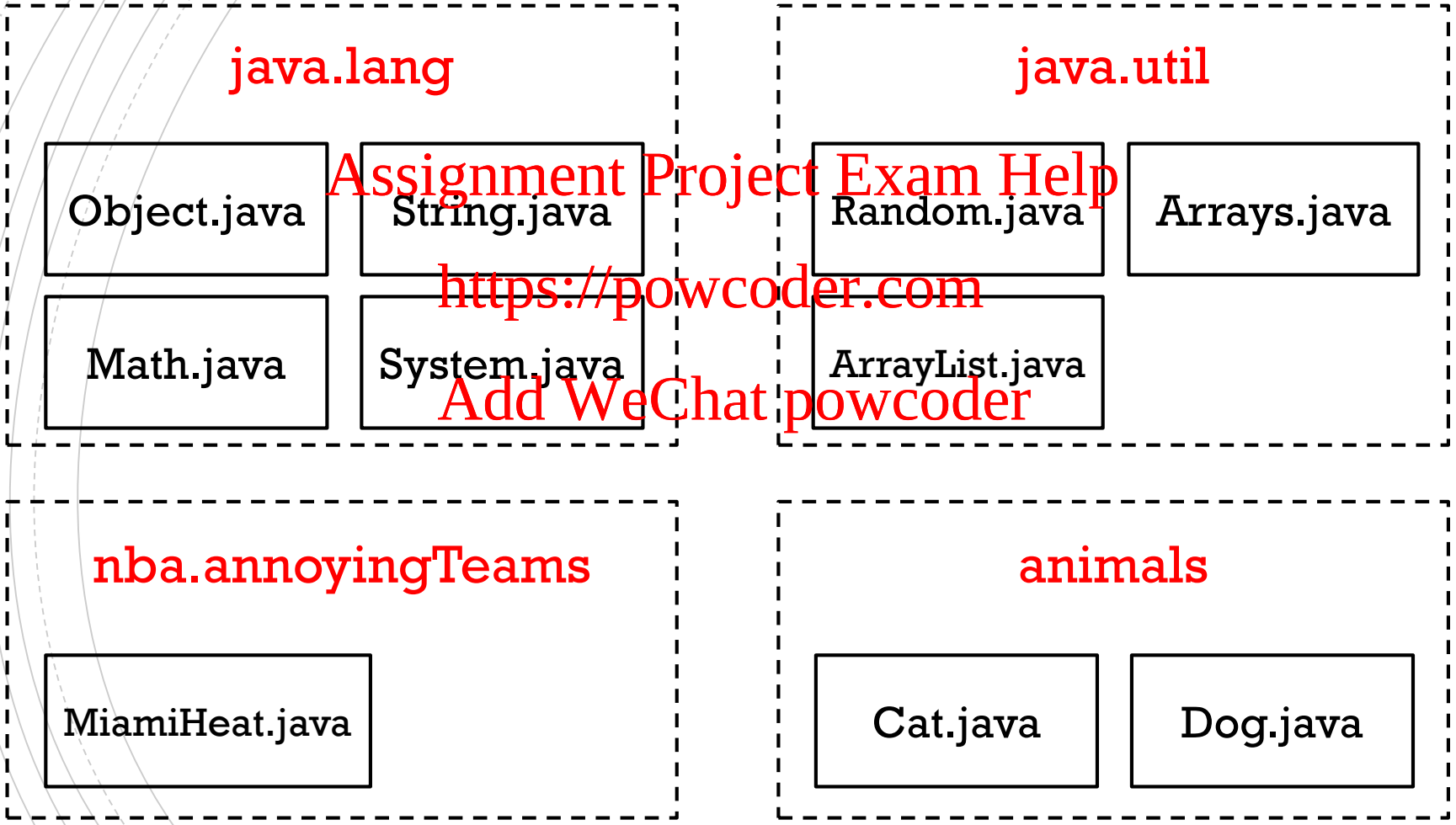
EXAMPLES



EXAMPLES



PACKAGES



USING A CLASS IN YOUR PROGRAM

If you want to use a *package member* from *outside* its package, you must instruct your program where to find that class. You can do this in 3 ways:

1. Specify the entire path whenever you use such class.
   For example, whenever you want to use `Dog` from the `animals` package you can *fully qualify* the class name: `animals.Dog`

```
animals.Dog myDog = new animals.Dog();
```

Ok for infrequent use!

USING A CLASS IN YOUR PROGRAM

If you want to use a *package member* from *outside* its package, you must instruct your program where to find that class. You can do this in 3 ways:

2. Import the package member. Example:

```
import animals.Dog;
```

This tells the computer that the class `Dog` is found in the package `animals`.

Ok if you use few members from a package.

USING A CLASS IN YOUR PROGRAM

If you want to use a *package member* from *outside* its package, you must instruct your program where to find that class. You can do this in 3 ways:

3. Import the entire package. Example:

```
import animals.*;
```

Now you can refer to any class inside the `animals` package.

USING A CLASS IN YOUR PROGRAM

For convenience, the Java compiler automatically imports two entire packages for each source file:

1. The `java.lang` package

2. The *current* package

This is why no import statement is need to use `Math`, `String`, … , or any *package member* from inside its own package.

INTRO TO OOP

RANDOM

How do you use it?

1. Import the corresponding class:
   - Identify the class in which we want to use it
   - Before the class definition, add the following statement:


```
import java.util.Random;
```

RANDOM

How do you use it?

2. Declare a variable of type `Random`, and create the object using the operator `new`.


```
Random randomGenerator = new Random();
Random otherGenerator = new Random(seed);
```

RANDOM

How do you use it?

2. Declare a variable of type `Random`, and create the object using the operator `new`.


```
Random randomGenerator = new Random();
Random otherGenerator = new Random(seed);
```

Declaration of two variables of type `Random`.

RANDOM

How do you use it?

2. Declare a variable of type `Random`, and create the object using the operator `new`.


```
Random randomGenerator = new Random();
Random otherGenerator = new Random(seed);
```

Declaration of two variables of type `Random`.

Creation of two `Random` **objects.**
Note: the result of the **`new`** operator is a ***reference*** to the new object.

RANDOM

How do you use it?

3. We called methods on the objects we created using the dot (.) operator


```
Random randomGenerator = new Random();
int randomNumber = randomGenerator.nextInt(100);
```

OBJECTS and CLASSES

OBJECTS

- An object is a collection of data and a set of methods can be provided to work with it. For example, a `String` is a collection of characters and methods like `charAt()` and `length()` can be used on it.
- Java is an object-oriented language. This means that it uses objects to represent data and provides methods related to them.
- Methods can take objects as parameters and produce objects as return values.
- Type of Objects we have seen up to now: `String`, arrays, `Random`.

IDEA: DEFINE YOUR OWN TYPE

- In Java, we can define our own type of data.
- The idea is that we can combine related pieces of information with each other into one variable.

HOW TO DEFINE A NEW TYPE

- When you define a class, you are actually defining a new type.
- So after defining `HelloWorld` you actually defined a type `HelloWorld`.
- This means that in another .java file, you actually could declare a variable of type `HelloWorld`.
- This wouldn't really make sense though because the new type doesn't *store* anything.

UP TO NOW

- Up to now we have created and used classes as containers for static methods. These kind of classes are called *Utility Classes*. An example of a utility class is the `Math` class.

- However, Java is an Object Oriented Programming language. In java classes can have a much bigger role!

CLASSES

By now, we should all know that objects and classes are closely related. How exactly?

- Each time we define a **class** we create a new **object type** with the same name.
- A class is a blueprint/template for a type of object. It specifies what properties the objects have and what methods can operate on them.
- An object is an **instance** of some class.

THE BLUEPRINT

```
public class ClassName {
   // some data declared here
   <modifier> <type> <variable_name>;

   public ClassName() {
      //constructor
   }

   // declare other methods
}
```


Data

Method to create an object

Other methods

File name: **ClassName**.java

NOTE ON NESTED CLASSES

- You can define a class *within* another class. We call such class a *nested class*.
  We refer to the class containing a nested class as the *outer class*.

- Why?
  - To group classes that are used only in one place.
    If a class is useful to only one class, it makes sense to keep it nested and together.

  - Increase encapsulation.
    Allows for better control over data.
  - Create readable and maintainable code.

FIELDS

STEP 1

```
public class ClassName {
   // some data declared here
   <modifier> <type> <variable_name>;

   public ClassName() {
      //constructor
   }

   // declare other methods
}
```


Data

Method to create an object

Other methods

File name: **ClassName**.java

STEP 1 – DATA

- Variables that denote the data stored by an object are usually called **fields** (or **attributes**).
- When defining a new data type the first thing to decide is the following:
  - What should be an attribute/field?
  - What type should these attributes/fields be?

SYNTAX

- fields are declared at the beginning of the class definition, outside of any method.

- Syntax:

```
<modifier> <type> <variable_name>;
```

```
<modifier> <type> <variable_name>;
```

Modifiers are **keyword** that you add to class/method/variable's definition to change their meaning. Java has different kind of modifiers, including:

Access Control Modifiers

- **`public`**
- `protected`
- *default* (no keyword)
- **`private`**

Non-Access Modifiers

- `static`
- `final`
- `abstract`

PRIVATE VS PUBLIC

- They are access control modifiers
  (keywords that determine from where a method or a variable can be accessed)
- `private`
  The method or variable that comes after is only accessible within the class in which it was written.


```
private int dontTouchMe;
```

- `public`
  The method or variable that comes after is accessible from anywhere

```
public int lookAtMe;
```

VISIBILITY/ACCESS CONTROL MODIFIERS

Note:
- outer classes can only be declared `public` or *package private*.
- members of a class (fields, methods, classes) can be declared using any of the access modifiers.

- public
- protected (= package + subclasses)
- default (= package)
- private

These modifiers define what is visible across classes.

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	Y/N	N
`private`	Y	N	N	N

DEMO

1. Create a class `Greetings` and write a method `hello()` that takes a `String` as input representing a name and displays an "`Hello <name>`".
2. Create a `Test` class and save the file in the same folder.
3. From within the `Test` class try to use the method `hello()` from the `Greetings` class.
4. Play around with the modifiers of the method `hello()` and see what happens.
5. In the `Greetings` class, add a field called `msg` and use it within or outside the class, while playing around with the modifiers.

Objectives:

- See the difference between public vs private
- Use a class we have defined from another class.

NON-ACCESS MODIFIERS

- **`static`**

  Fields, methods, and nested classes can be declared to be `static`.

When a class member is declared to be static, then it "belongs" to the entire class and not to a specific instance (object).

- **`final`**

  Variables, methods, and classes can be declared to be `final`.

- **`abstract`**

  Methods and classes can be declared to be `abstract`.

STATIC

- We can define a field or a method to be `static` if we want it to be independent from one specific instance of the class.

- A static method/field is associated *with the entire class*
  Static fields are also called ***class variables***.

- A non-static method/field belongs to *an instance of the class*
  Non-static fields are also called ***instance variables***.

STATIC VS NON-STATIC

```
String s = "hippos";
String t = "elephants";
boolean b = (s.length() == t.length());
```


`length()` **is non-static method. Its execution depends on a specific string.**

STATIC VS NON-STATIC

```
double x = Math.PI;

int y = Integer.parseInt("1");
```


- `PI` **is a static field. It belongs to the** `Math` **class.**
- `parseInt()` **is a static method. It belongs to the** `Integer` **class and does not depend on a specific object of type** `Integer`**.**

STATIC VS NON-STATIC

	static	non-static
Associated with	entire class (one per class)	instance of a class (one per object/instance)
How to call (methods) from outside the class	ClassName.methodName()	obj.methodName()
How to reference (data) from outside the class	ClassName.varName	obj.varName

DEMO

Go back to the `Greetings` class and now play around with the static modifier. Both with the method and with the field.

LOCAL VARIABLES VS FIELDS

How do they differ?

- Where to declare them:
  - Local variables are declared inside a method or a block

  - Fields (class and instance variables) are declared inside a class, but outside a method

LOCAL VARIABLES VS FIELDS

How do they differ?

- Scope:
  where can they be accessed (called directly using the variable name)
  - Local variables can be accessed only within the method or block in which they have been declared.
  - class variables be accessed from any method or block in that class
  - instance variables can be accessed from within the class or from non static methods of the class

LOCAL VARIABLES VS FIELDS

How do they differ?

- Access:
  - Local variables cannot have access modifiers. You can't access local variables from other classes or methods.
  - Field can have access modifiers. They can be accessed from methods within the class and from other classes if declared `public`.

http://edayan.info/java/fields-vs-variables-in-java

EXAMPLE - Student

- Useful Data:
  - Name
  - Student ID
  - Grades
  - Courses Taken
- By itself, this code is a legal class definition
- Note:
  - The class is `public`, it can be used by other classes
  - The instance variables (non-static fields) are `private`, they can only be access from inside the class. If you try to access them from other classes you will get a compile-time error.

```
public class Student{
    private String name;
    private int studentID;
    private double[] grades;
    private String[] courses;
}
```

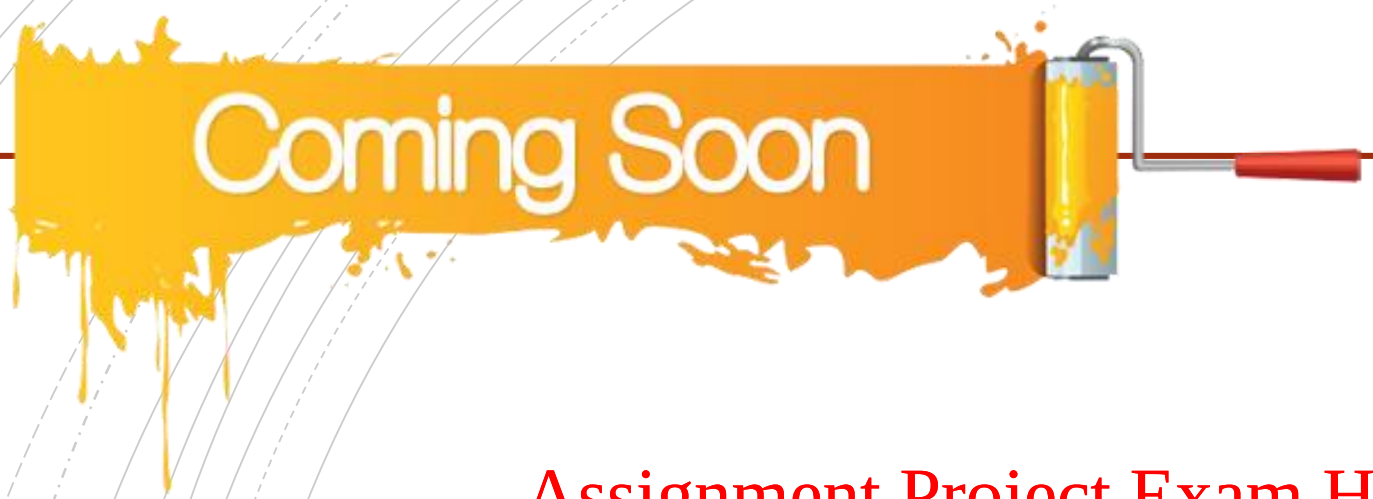
TRY IT!

How to use our new type `Student`:

- The snippet of code from before must go into a class called *Student.java*
- Now, create a second file *TestStudent.java* that goes inside the same folder as *Student.java.*
- Create a main method in the *TestStudent* class.
- Try to declare and initialize variables of type *Student* both from within the *Student* class and the *TestStudent* class.

TO LOOK FORWARD TO

- We'll talk more about `final` in a couple of videos as well as after learning about inheritance next week.
- In a week, we will also learn about `abstract` classes and methods.



In the next video:

- Constructors
- `this`