



ORACLE

Academy



Oracle Academy

Java for AP Computer Science A

7-5

Object Interaction and Encapsulation

ORACLE
Academy



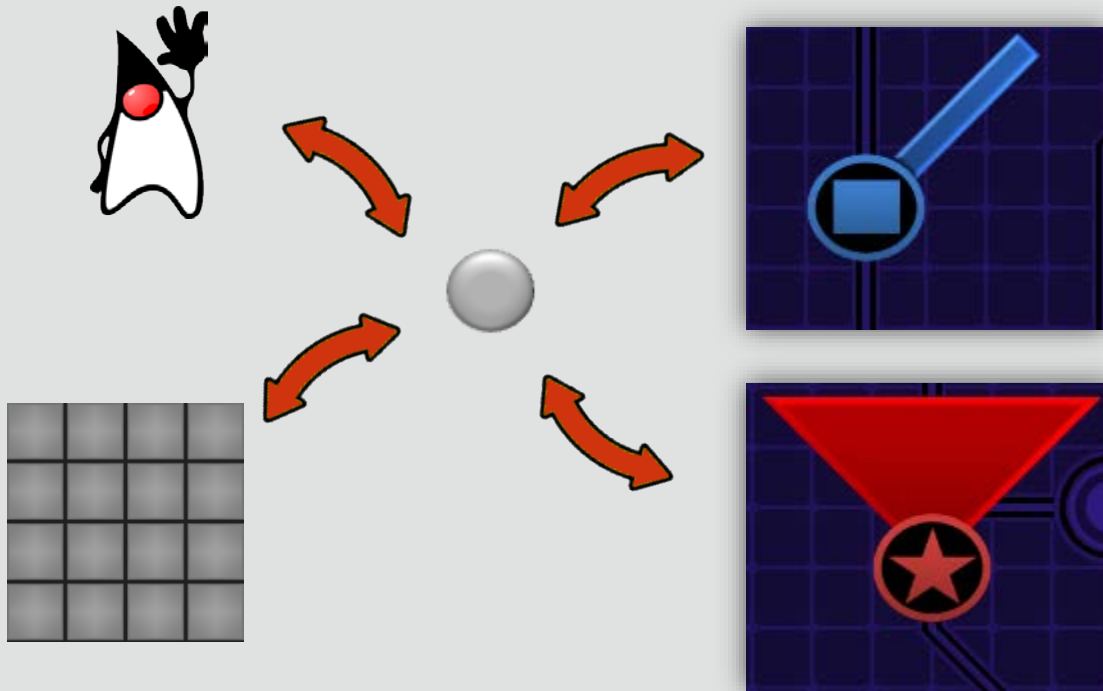
Objectives

- This lesson covers the following objectives:
 - Understand object interaction in greater detail
 - Use the private modifier to define class variables
 - Understand the purpose of getter methods
 - Understand the purpose of setter methods



Object Interaction

- Section 2 introduced the idea of object interaction
 - No prescribed sequence for how object must interact
- This lesson explores how to program interactions



What Is Object Interaction?

- An object reference is a memory address
 - A reference directs one object to another
 - A reference allows one object to interact with another
- Objects interact by ...
 - Accessing another object's fields
 - Calling another object's methods
- If the main method instantiates every object ...
 - The main method contains every object reference
 - The main method can access every objects' fields and methods

Example Program

- Consider a program that models Prisoner, Cell, and Guard objects
- The main method may look like this:

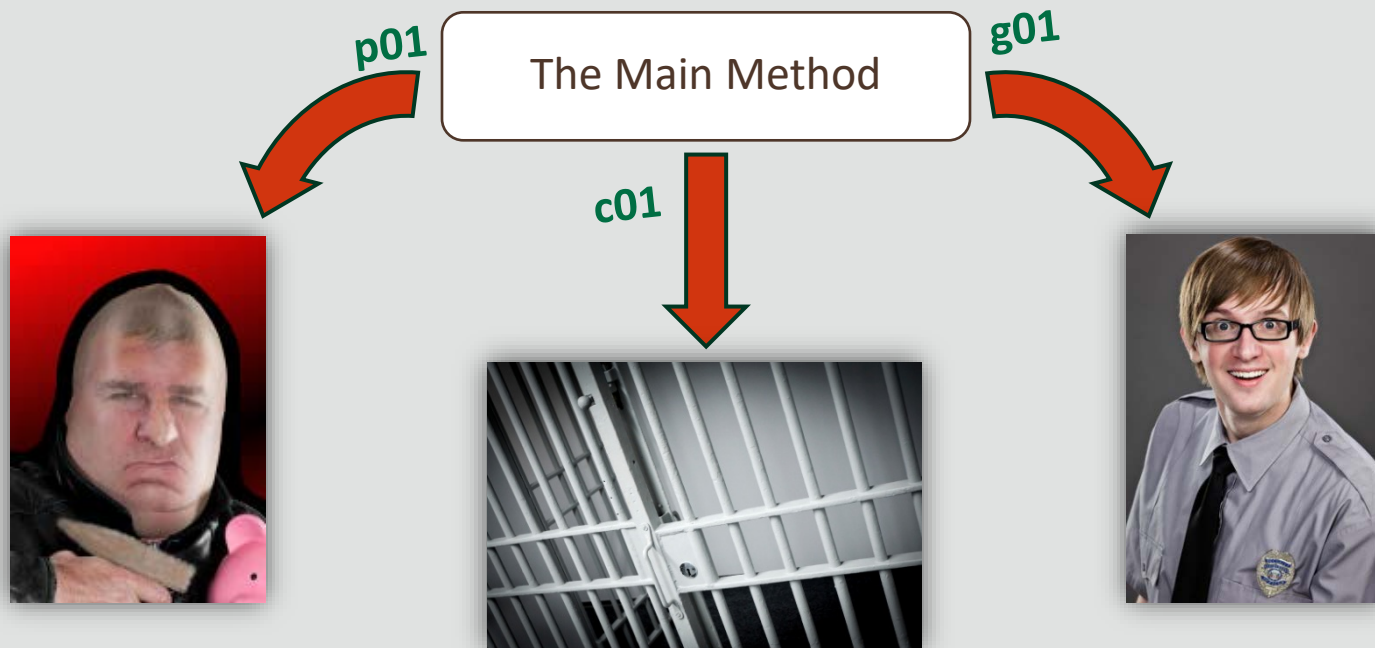
```
public class PrisonTest{  
    public static void main(String[] args){  
        Prison p01 = new Prisoner();  
        Cell c01 = new Cell();  
        Guard g01 = new Guard();  
  
        p01.name = "Bubba";  
        c01.name = "A1";  
        g01.name = "Boss Man";  
    } //end method main  
} //end class PrisonTest
```

Object references

Interactions

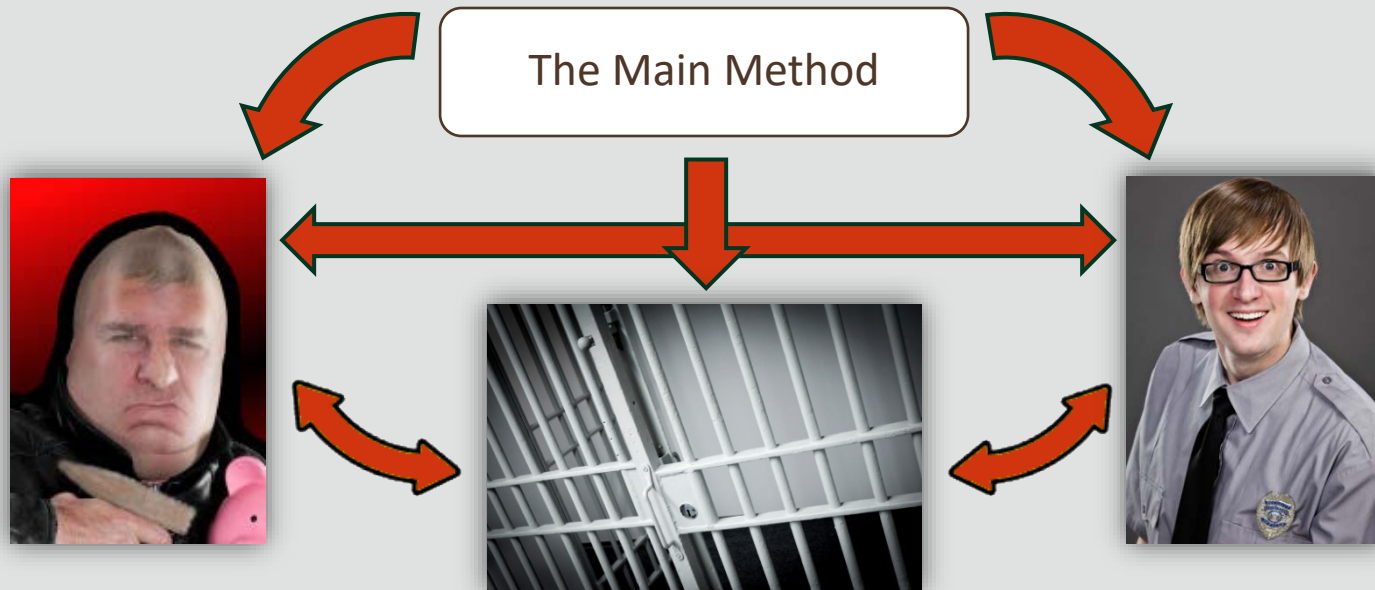
Interactions from the Main Method

- The main method contains all the object references
- Therefore, the main method controls all interactions in this system



Interactions Between Objects

- However, sometimes you'll want a program where objects interact with each other
- To do this, objects must know about each other
 - One object must know a reference to the other object



How Do Objects Know About Each Other?

- Object references must be shared:
 - One object may contain another object as a field
 - One object's method may accept another object as an argument
- For example:
 - A way to describe a Prisoner is by their Cell number
 - It could be argued that a Cell is a property of a Prisoner object
 - The Prisoner class would contain a Cell field

Exercise 1, Part 1

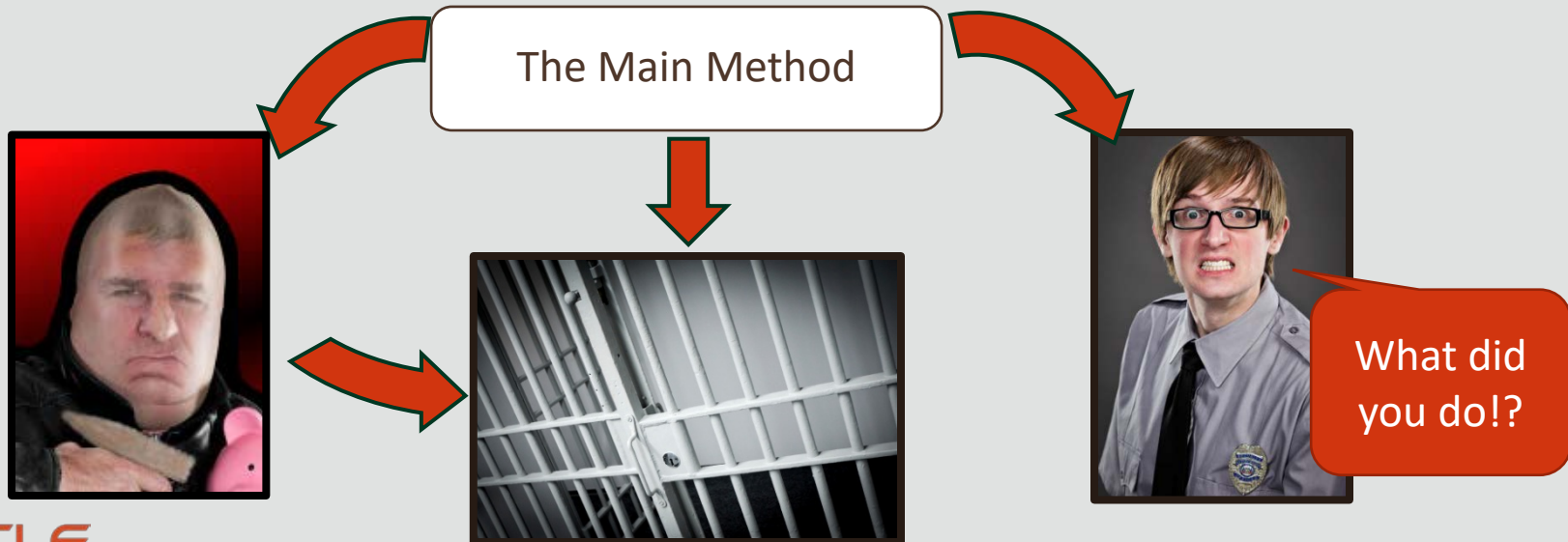
- Continue editing the `PrisonTest` project
 - A version of this program is provided for you in the files `PrisonTest_Student_7_5.java` and `Prisoner_Student_7_5.java`
- Create a `Cell` class that includes the following:
 - String name of the cell
 - Boolean describing whether the door is open
 - Two-argument constructor that sets both fields
- Modify the `Prisoner` class so that it:
 - Includes a `Cell` field
 - Sets the `Cell` field based on a constructor parameter
 - Prints the cell's name as part of the `display()` method

Exercise 1, Part 2

- Write an `openDoor()` method in the `Prisoner` class
 - Access and modify the corresponding field in the `Cell` object so that:
 - If the door is closed, open it
 - If the door is open, close it
 - Print whether the door opens or closes
- From the main method:
 - Instantiate a `Cell` and a `Prisoner` object
 - Call the prisoner's `display()` method once
 - Call the `openDoor()` method a few times

Oops!

- The guards are panicking!
- Your program allows prisoners to access their cell doors
- Considering Bubba's plans for revenge, this type of interaction should not be allowed!



Think About Potential Object Interactions

- Consider which objects must know about each other
 - Some objects have no business modifying another's fields
 - Try to minimize their knowledge of each other ...
 - This prevents unwanted results and make code less complicated
- Consider which direction the interactions might occur and which objects should be properties of each other
 - Should a Prisoner have a Cell property?
 - Should a Cell have a Prisoner property?
 - Or should neither know about each other?

Think About How to Distribute Behaviors

- Cells are designed to open and close
 - Someone must have access to perform these interactions
 - Prisoners should not be capable of this behavior
 - Guards should be capable of this behavior
- Deciding how to distribute behaviors between objects is an important challenge of object-oriented programming
 - But don't worry. You already have experience doing this
 - A major goal of Java Puzzle Ball was to create situations where players needed to think carefully about distributing behaviors between different object types

Introducing Encapsulation

- Sometimes objects must know about each other
- Encapsulation provides techniques for limiting one class's visibility of another
- It's possible to restrict which fields and methods other classes can see
- Special methods can be written to decide how data should be accessed and modified
- Access and visibility should be limited as much as possible

Access Modifiers

- The public keyword is one of several access modifiers
- Access modifiers limit the visibility of fields and methods between classes

```
public class Cell {  
    //Fields  
    public String name;  
    public boolean isOpen ;  
  
    //Constructor  
    public Cell(String name, boolean isOpen){  
        this.name = name;  
        this.isOpen = isOpen;  
    } //end constructor  
} //end class Cell
```

Access Modifier Details

- **public:** Visible to any class
 - It's the least secure
 - Methods are typically public
- **Package:** Visible to the current package
 - There's no keyword for this level of access
- **private:** Visible only to the current class
 - It's the most secure
 - Fields are typically private

Exercise 2

- Continue editing the `PrisonTest` project
- Modify the `Cell` class:
 - Change its fields to private
 - Save the file
- Does your IDE have any complaints?
 - What are the complaints?
 - Where do they occur?

The Effects of Private Data

- The following private fields can't be accessed outside the Cell class:
 - isOpen
 - name
- Even the main method can't access this data
- It's good that prisoners can't open their cell doors
- It's bad that prisoners don't know the names of their cells
 - The next topic discusses how to address this issue

Introducing Getter Methods

- When a field is inaccessible, it can't be:
 - Read
 - Modified
- However, in many cases it's desirable for one class to at least know the value of another class's fields
 - A prisoner should at least know their cell name
 - This requires a prisoner to read the value of a Cell's name field
- Getter methods provide a solution

Getter Methods

- Getters are also called accessors
- Getters are public
- Getters usually accept no arguments
- Getters return the value of a particular variable
 - Most private variables require a getter method

```
public class Cell {  
    ...  
    public String getName(){  
        return name;  
    }//end method getName  
    public boolean getIsOpen(){  
        return isOpen;  
    }//end method getIsOpen  
}//end class Cell
```

Introducing Setter Methods

- In other cases, it's desirable for one class to modify another class's field
- However, this must be done safely
- For example:
 - A guard should be able to open a door, but a prisoner should not
 - A bank account balance should not drop below zero
- Setter methods provide a solution

Setter Methods

- Setters are also called mutators
- Setters are usually public
- Setters usually accept arguments
- Setters are void type methods

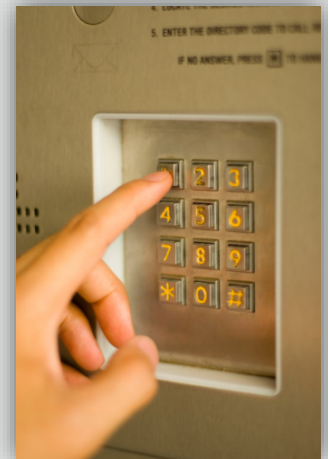
```
public class Cell {  
    ...  
    public void setName(String name){  
        this.name = name;  
    }//end method setName  
    public void setIsOpen(boolean isOpen){  
        this.isOpen = isOpen;  
    }//end method setIsOpen  
}//end class Cell
```

Designing Setters

- Be careful when you write setters like those shown on the previous slide
 - Prisoners would again have access to their doors
- Sometimes a little thought needs to go into designing a setter method
 - A security door may ask for a security code
 - Banking software may check whether a withdrawal amount would result in a balance less than zero or if the withdrawal amount is negative

Exercise 3, Part 1

- Continue editing the `PrisonTest` project
- Modify the `Cell` class so that ...
 - Getters exist for the `name` and `isOpen` fields
 - There's a private 4-digit security code field, it's initialized from the constructor and has no getter method
 - There's a setter for opening/closing the door, and it does the following:
 - Accepts a security code as an argument
 - Prints if the code is incorrect
 - If the code is correct and the door is closed, opens it
 - If the code is correct and the door is open, closes it
 - Prints if the door is opened or closed



Exercise 3, Part 2

- Modify the `Prisoner` class so that ...
 - The `display()` method prints the cell name
 - The `openDoor()` method is removed
- Modify the main method so that ...
 - The `Cell` is instantiated properly
 - The prisoner no longer tries to open the cell door
 - It tests a cell class's ability to open and close its door
 - Try supplying both correct and incorrect security codes



Continuing to Develop This Software

- Currently, the main method tests a Cell door's ability to open and close based on a security code
- Testing allows us to confirm that this feature is implemented properly
 - If the feature doesn't work, it should be fixed
 - If the feature does work, it's safe to include this feature as part of another feature
- A possible next step would be to develop a Guard class with a method for inputting a security code
 - Ultimately a guard, not the main method, would be responsible for inputting a security code

The Role of the Main Method

- Some programs are driven by physical objects
- Some programs are driven by buttons
- In this exercise, the main method models actions that would drive the program
 - Calling `bubba.openDoor()` models a prisoner trying to open their cell door
 - Calling `cellA1.setIsOpen(1234)` models a person who entered a security code

Exercise 4

- Continue editing the `PrisonTest` project
- Encapsulate the `Prisoner` class
 - Make its fields private
 - Provide getters and setters for every field

That Exercise Wasn't Fun!

- Was Exercise 4 tedious and did it make you groan?
- Some programmers prefer the control of encapsulating fields themselves
- Other programmers would rather have their IDE do the work for them
 - Your IDE can encapsulate fields for you
 - The following slides show how to do this in NetBeans
 - If you are using a different IDE, consult the documentation for how to do this



NetBeans Encapsulation Trick

1. Highlight the fields that you want to encapsulate

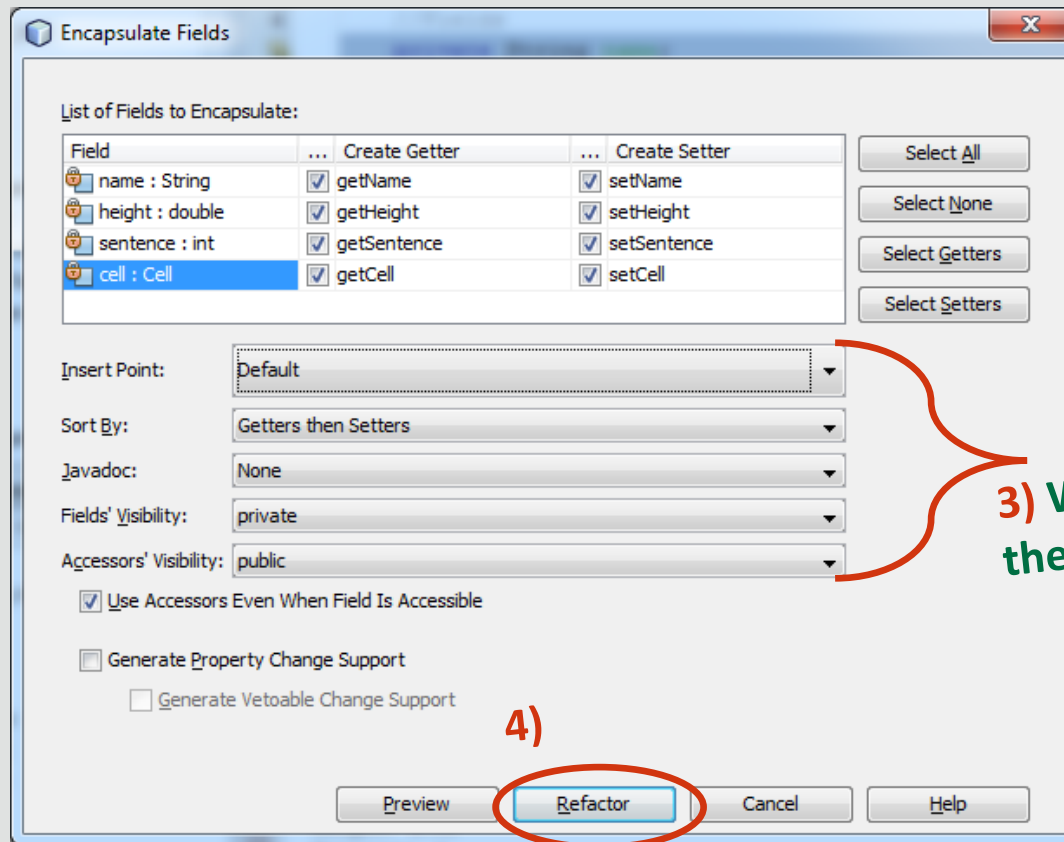
```
3 public class Prisoner {  
4     //Fields  
5     public String name;  
6     public double height;  
7     public int sentence;  
8     public Cell cell;  
9 }
```

2. Right-click and select **Refactor >> Encapsulate Fields**

NetBeans Encapsulation Trick

3. Adjust the settings as you like

4. Click Refactor



3) We recommend these settings

4)

Summary of Encapsulation

- Encapsulation offers techniques for limiting the visibility of a class
- Access and visibility should be limited as much as possible
- Most fields should be private
- Provide getter methods to return the value of fields
- Provide setter methods to safely modify fields

Summary

- In this lesson, you should have learned how to:
 - Understand object interaction in greater detail
 - Use the private modifier to define class variables
 - Understand the purpose of getter methods
 - Understand the purpose of setter methods





ORACLE

Academy

