ALGONQUIN COLLEGE

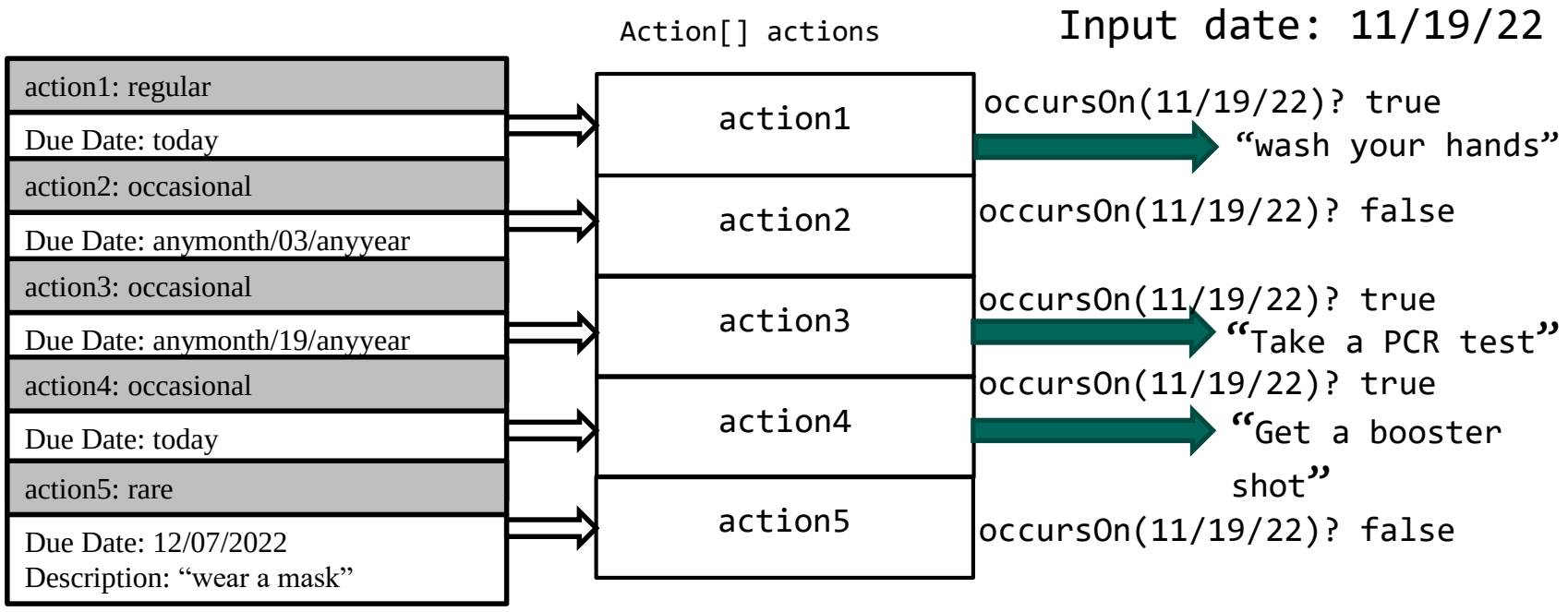
CST8284

SOME ADDITIONAL INFORMATION ON

ASSIGNMENT 1

CST8284 Assignment 1 Requirements

In Assignment 1, you will instantiate various subclasses of the Action class, and store each instance in an array. In Part 3 of the assignment you'll search the array to see if the date of each Action object matches the date requested by the user, i.e. which action task is 'occursOn' that particular date. Whenever it is, that object's description is printed out using its toString() method.



CST8284 Assignment 1 Requirements

Assignment 1 consists of three parts, and only the third part is responsible for demonstrating the full capabilities of your program, indicated above.

Part 1 requires that you add an abstract occursOn() method to the **Action** class. As an abstract method, it will have *no body, just the header*. And because we may wish to have actions for one or more days of the year, occursOn() needs to take three parameters: ints for day, month, and year. Action also needs a one-arg constructor to load the description String. (Ideally, as a superclass, it should have a no-arg constructor as well, although this is not absolutely essential here.)

The **RegularAction** class extends Action, and should include its own one-arg constructor chained to the one-arg superclass constructor in Action. Furthermore, since Action contains an abstract method, all its subclasses will need to override occursOn() with a concrete implementation of that method—including RegularAction.

occursOn() determines if there are any tasks to report for this day, depending on the type of Action subclass. For example, if something occurs every day, then the actual date passed doesn't really matter: any day you enter matches the requested date, since that task happens *every day*.

CST8284 Assignment 1 Requirements

Now create a class called **ActionDriver** to test your new subclass:

- Test that RegularAction is a subclass of Action

  Output a string to indicate the type of test—"RegularAction is just a subclass of Action: "—followed by the java reflection code needed to compare RegularAction's superclass with the Action class itself.

  The expected output is 'true'. So after you've compared RegularAction's superclass with the Action class itself (using '=='), output a line to remind the user that the expected output is true.

```
RegularAction is just a subclass of Action: true
Expected: true
```

CST8284 Assignment 1 Requirements (2)

- Since the field(s) used by RegularAction is inherited from its superclass, test that RegularAction has no declared fields of its own.

  Again, output a String that indicates the test for extra fields in RegularAction, run the appropriate java reflection test, and indicate the expected result.

```
RegularAction activities have no extra fields:true
Expected: true
```

- Instantiate a new RegularAction constructor and load it with the message indicated in the sample output. Having loaded an item, you should now be able to output it's description using the object's toString() method. Then output the result of occursOn() and the expected output.

```
Looking at regular actions: Wash your hands
Expected: Wash your hands
true
Expected: true
```

CST8284 Assignment 1 Requirements (3)

In **Part 2** you add two more subclasses, **OccasionalAction** and **RareAction**, for items that occur on the same day each month, and only once every year, respectively. Again, the shell of these two class is provided for you, and you must override the occursOn() method so that it returns true or false if the new subclass object's day, month, and year meet the requirement specified for that class.

Again, these two new classes should have *appropriate* constructors and overridden concrete implementations of occursOn(), which returns an appropriate boolean value depending on the requirements of the subclass.

Create a new test class, ActionDriver2, that includes tests for the following:

- Test that both test classes have the same superclass, Action
- Test the number of fields for each subclass. For the **OccasionalAction** class, only the day of the month is required for a comparison. For the **RareAction** class, the day and month are required, hence its constructor would set three fields.
- Again, instantiate new instances of both subclasses, load a description, and test the occursOn() method in each.

CST8284 Assignment 1 Requirements (4)

In **Part 3** you'll instantiate different Action objects, load them into an array, and perform the tests indicated by the sample output—as described in the first slide.

- Load an array of Action objects of various types, i.e. Regular, Occasional, and Rare. Create various different descriptions of the tasks to be performed.
- Instantiate a new Scanner, prompt the user to enter a date as three ints corresponding to year, month, day (clearly indicating the order in which these values are entered)
- Loop through the array of Action objects, test each object's occursOn() method against the three int's just entered by the user, and then, whenever the method returns true, print that object's toString() method, displaying the task stored in the object's description field.

CST8284 Assignment 1 Notes

Notes:

- You cannot use an external class, like Calendar or Date, to perform the operations that your code is supposed to do

- All your program code should be in a package named

```
assignment1.Action.secXXX
```

where XXX corresponds to your lab section number.

Remember to change default package name ‘student1’ to the package name specified above.

- You must provide a UML diagram with your upload. For your diagram you can drag and drop it with your program zip file to the upload link. Do not use auto-generating apps. You can use UMLet or MS Word tools.

CST8284 Assignment 1 Notes (2)

Notes (con't):

5. There are no JUnit tests involved in this Assignment
6. Each method, field, and class must be thoroughly documented
7. Your upload must include a .doc folder containing an index.html file that correctly displays your Javadoc comments. Only the comments that appear in hypertext-linked form in the doc folder are marked. So check to make sure that Javadoc compiled your comments correctly and completely.
8. Use proper indentation in your classes to indicate, for example separate methods, code blocks, etc.

These additional notes have been contributed by Prof. David Houtman.