# YOUR PROJECT TITLE AND ACRONYM REPLACE THIS

e.g., PANTRACK: A System for Tracking Pandemics in Iceland

# YOUR NAME REPLACES THIS

# Assignment 1 --1/12/2021 7:08 AM

For this assignment, you will describe and implement release 1 of your term project. You will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*. You are free to choose a project that interests you but if you prefer, the instructor and your facilitator will be happy to suggest a topic. If you are already an experienced developer, this is an opportunity to build a challenging application (check with your facilitator if it requires significant API’s) or discuss research with the instructor. It is OK to name a project with much more scope than you can accomplish in the course (as in the example above): we will not hold you to completing everything associated with it. What we do expect is that you specify and implement a set of do-able requirements within such scope.

Submit this completed Word document. Replace as indicated. Please observe and retain the gray text. Your materials—in black 12-point Times New Roman—should not exceed 5 pages excluding references, figures, and appendices. Use the Appendix sections for additional material if you need to. These will be read only on an as-needed basis.

We want you to develop in Eclipse preferably or else IntelliJ (talk to your facilitator about exceptions). As you code, use JUnit tests whenever possible but certainly by week 2—package-by-package, class-by-class, and method-by-method, except for trivial methods and those requiring I/O. Use non-Junit classes for testing the latter. Keep the evaluation criteria in mind, listed at the end.

For this assignment, you do not need to read data from a file—you can build all data into the code.

Include a ReadMe file describing where to run the application from, and including necessary execution notes. All JUnit tests will be assumed runnable.

# 1.1 SUMMARY DESCRIPTION *EVALUATION CRITERION (i) APPLIES*

One- or two-paragraph overall description of your proposed term project—half-page (12-point Times New Roman) limit. By the end, term projects will incorporate most of the techniques discussed in the course. To do this, you may need to alter the direction of your project or introduce an additional project in future. You may alter this or even replace it as the semester progresses. You will probably find it useful to use your project acronym.

Your response should replace this sentence, as well as the example below.

This project concerns a system for students to learn written material that is between a paragraph and several pages in length. To use the system, called *Knowla*, students respond to a question by selecting from and ordering a set of fragments (typically sentences). Since material that’s more than a page in length contains too many sentences to re-order, we envision a system that first requires ordering paragraph topics, and after that the sentences within each paragraph. The focus of the term project is the automatic grading of solutions to Knowla tests. This grading system will be called *K-Grade*.

## 1.2 I/O EXAMPLE FROM *PROJECTED* COMPLETED PROJECT *EVALUATION CRITERION (i) APPLIES*

Provide an example of projected *concrete* output for designated input. You will not be held to fulfilling exactly this—it is just explanatory at this point, to indicate where your project is going. We recognize that project direction and details will change as the term progress. This section refers to the project as a whole, not just to what you will produce this week, so we can gain an idea of what you have in mind overall.

Your response should replace this sentence, as well as the example below.

Here is an example of a simple Knowla test.

**Knowla**:

Reconstruct “Humpty Dumpty” from the following set of fragments, inserting punctuation and capitalization. You need to provide only the correct order, e.g., 5, 3, 1, 2

1. all the king's horses and all the king's men
2. to glue him back up
3. couldn't put Humpty together again
4. Humpty Dumpty sat on a wall
5. never were able
6. Humpty Dumpty had a great fall

**User**: 4, 5, 2

**System**:

Your answer is:

Humpty Dumpty sat on a wall.

Never were able

to glue him back up.

You are correct that line 1 should be fragment 4 (+30%).

Even though “never were able” and “to glue him back up” are not in the poem, you get credit for ordering them correctly (+2%).

Fragment 5 (“never were able”) is not part of the poem. (It is similar to part of the real third line “couldn't put Humpty together again.”) (-1%)

Fragment 2 (“to glue him back up”) is not part of the poem. (It is similar to part of the real third line “couldn't put Humpty together again.”) One clue is that “up” does not rhyme with any other ending. (-1%)

Fragment 1, “All the king's horses and all the king's men,” should be line 3 in the poem, following “Humpty Dumpty had a great fall.” (-2%)

Fragment 3, “couldn't put Humpty together again.” should be line 4 in the poem, following “all the king's horses and all the king's men.” (-2%)

Your grade is 26%.

## 1.3 REQUIREMENTS IMPLEMENTED IN THIS RELEASE *EVALUATION CRITERION (ii) APPLIES*

Supply [functional requirements](https://docs.google.com/document/d/1eU7eINLDxmrf793D4OF2yGT4ry_SW3GQGoVDYzecGHc/edit?usp=sharing) statement that you accomplished for this assignment, i.e., functionality that the application provides for the user. Please state requirement in declarative form, as illustrated in the examples, because here we want to know the functionality intended (*what*, not *how*). For example, the following is *not* a proper functional requirement: *TicTac will have a class for O’s and a class for X’s.* It is common to mistake design elements like this for requirements. To get started, state what the application will accept as input, like requirement 1.3.1 below.

Keep in mind that the implementation of your requirements will incorporate *an abstract class*, *inheritance*, *upcasting or downcasting*, and *polymorphism*; that will probably influence the requirements you choose to implement in this assignment. The example material supplied should be deleted before you submit.

### 1.3.1 <Your requirement title replaces this> Create Hierarchies

Your first requirement should replace this, as well as the example below

Knowla shall accept a hierarchy of heading and non-heading fragments (but currently in hard-coded form only).

### 1.3.2 <Your requirement title replaces this > Report Fragment Hierarchies

Your response should replace this, as well as the example below

Knowla shall report all of the heading fragments it possesses with their subheadings to the console.

## 1.4 ILLUSTRATIVE OUTPUT FROM IMPLEMENTATION *EVALUATION CRITERION (ii) APPLIES*

### Provide illustrative output from your implemented application (so far) showing that the requirements have been met. Explain what class.method(s) produce it.

Your response should replace this, as well as the example below.

The following is produced by *FragmentDisplayExample.displayFragmentHierarchyExample()*. *HeadingFragment* and *NonHeadingFragment* objects are hard-coded data.

=========FragmentDisplayExample==========

========= Should display 8 items ==========

My Expenses Heading

Taxes Heading

Estate taxes

Income taxes

Housing Heading

Mortgage for housing

Maintenance for housing

Food

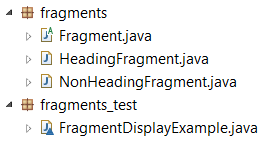
3 headings and 5 others.

## 1.5 YOUR DIRECTORY

### Show a screenshot of your directory. This should include a parallel directory of JUnit tests where possible—package-by-package, class-by-class, and method-by-method, except for trivial ones.

Your response should replace this, as well as the example below.

All methods in the K-Grade example are trivial so far except for ones that perform displays, and these are not suitable for JUnit testing.



## 1.6 TECHNIQUES IMPLEMENTED *EVALUATION CRITERION (iii) APPLIES*

Your implementation should include *inheritance*, *polymorphism*, and *either an abstract class or interface* at least once, and in a manner that is useful to your application. Explain where and how you applied these, using the headings below.

### 1.6.1 Class model and Sequence Diagram

Identify where you included *inheritance*, *polymorphism*, and *abstract classes* or *interfaces* in your class model. Make classes and members *static* or not as per their intended usage. To do this use tools (e.g., Visio and Lucidchart), PowerPoint, or a combine models as in [this example](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing) (which you are free to cut and paste from). Insert indications in red (as in [this example](https://docs.google.com/spreadsheets/d/1ZvkerE9FkWHWwVGdzuy7YMBU6oBMFGZbA4sotFETs8Y/edit?usp=sharing)) to show where the three features below apply.

Your response should replace this, as well as the example below.

The class model for the K-Grade example is [here](https://docs.google.com/spreadsheets/d/1vBmDVtWWh3EX0oehFFLRU0P6eR-fn4d0qVg1-XOUooM/edit?usp=sharing). *Fragment* is abstract. Polymorphism occurs in the versions of *getType*(). The figure includes inheritance.

### 1.6.2 Code showing an abstract class or interface

Show the relevant code (only) and explain why an abstract class or interface is appropriate here. It should be clear where the code is located (class and method).

Your response should replace this, as well as the example below.

There are only two kinds of fragments, so there is no need for the *Fragment* base class to be concrete since there will be no generic Fragments.

**package** fragments;

// A portion of a text passage, such as a sentence

**public** **abstract** **class** Fragment {

// Pedagogical: (1) Attributes (2) Constructors (3) Methods,

// alphabetically ordered within each.

**protected** String text = "text not determined yet";

**public** Fragment() {

}

**public** Fragment(String someText) {

text = someText;

}

// Pedagogical: (expressive) names alone are often sufficient for abstract methods.

**public** **abstract** **void** display();

**public** **abstract** String getText(); // Pedagogical: "get..()" and "set...()" standard

**public** **abstract** String getType();

**public** **abstract** **void** setText(String someText);

}

### 1.6.3 Code showing polymorphism

Show the relevant code (only) and explain why *polymorphism* is appropriate here. Recall that polymorphism is implemented in one of two ways – overriding methods in subclasses or overloading methods in the same class where the method signatures are different – and allowing the language runtime to dynamically invoke the correct method. It should be clear where the code is located (class and method).

Your response should replace this, as well as the example below.

The method *displayTypeCount*() iterates through Fragment’s and calls *getType*() on each. This is polymorphism. The relevant code is shown below.

**static** **void** displayTypeCount() {

// Postcondition: The counts of heading- and non-heading type fragments in

// theFragments are on the console

**int** headingCount = 0, nonHeadingCount = 0;

**for** (Fragment fragment : *theFragments*) {

**if** (fragment.getType() == "Heading fragment") {

headingCount += 1;

}

**else** {

nonHeadingCount += 1;

}

}

…

}

### 1.6.4 Code showing upcasting or downcasting

Show the relevant code (only) and explain why upcasting or downcasting is appropriate here. It should be clear where the code is located (class and method).

Your response should replace this, as well as the example below

Some of the sub-fragments of a *HeadingFragment* are themselves *HeadingFragment*s. We need to cycle through all sub-fragments but identify *HeadingFragment*s so we can call *displayAll*(). Downcasting is needed from *Fragment* to *HeadingFragment*.

**public** **class** HeadingFragment **extends** Fragment {

**private** ArrayList<Fragment> theSubFragments = **new** ArrayList<Fragment>();

**….**

**public** **void** display() {

System.***out***.println(text);

}

**public** **void** displayAll() {

display(); // the heading itself

// display sub-headings and their sub-headings

**for**(Fragment fragment: theSubFragments) {

**if** (fragment **instanceof** HeadingFragment) {

((HeadingFragment)fragment).displayAll();

}

**else** {

fragment.display();

}

}

}

## 1.7 YOUR CODE

Unless your facilitator requests another method, copy your Eclipse project to your file system, zip it, and attach it. Please contact your facilitator in advance if you want to request an alternative means.

<Your response here>

## 1.8 EVALUATION OF ASSIGNMENT 1



## Appendix 1 (if needed; should be referenced above, and will be read as-needed only)

## Appendix 2 (if needed; should be referenced above, and will be read as-needed only)