**Math & Computer Science – Winter 2016**

**Case Study – Algorithm Cover – 420-204-RE**

Team members: **Deadline: 19-February-2016**

|  |  |  |
| --- | --- | --- |
| Li, An | Patel, Sagar | Sevastis, Dimitri |
| Student 1 (last name, first name) | Student 2 (last name, first name) | Student 3 (last name, first name) |

Use Word to fill in these boxes to provide **an estimate of the total number of these constructs** appearing on the algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| Beans | 3 | Properties | 37 |
| Interfaces | 1 | Abstract classes | 1 |
| Protected members | 62 | Private members | 37 |

Fill in the table below by specifying the animating formulas and properties (if any)

|  |  |  |
| --- | --- | --- |
| **Physics topic** | **Formula 1** | **Formula 2** |
| **Mechanics (203-NYA)** |  |  |
| **Electricity & Magnetism**  **(203-NYB)** | Fe = | V = R × I |
| **Waves & Modern Physics**  **(203-NYC)** | y(t) = Asin(ωt + ϕ0) | asinθ = mλ |

Briefly explain why **efficiency** should be considered for grading by indicating the formula involved and by describing the specific efficiency-related task in the algorithm.

|  |
| --- |
| Efficiency should be considered because it uses less memory, is easier to understand and requires a good understanding of the programming language to achieve.  Example: Projectile motion formulas including because it makes use of constants that have already been declared in the interface (zero, acceleration due to gravity), uses methods/abstract methods that have been previously declared (since they are reused several times), and because there is a consistency between the button methods between topic classes (i.e.: all buttons do the same between classes, but with different objects involved). |

Briefly explain why **complexity** should be considered for grading by indicating the formula involved and by describing the specific complexity-related task in the algorithm.

|  |
| --- |
| Complexity should be considered because it gives a certain difficulty in the coding of the program.  Example: Kinematics formulas including because it contains many variables, including breaking vectors in x and y components, so multiple methods should be used to compute the velocities’ x and y components, magnitude and angle, time, displacement, height over time and velocity magnitude over time. |

**Others**:

* Submit a hierarchical organization of the classes appearing on this algorithm
* Do not submit any Java statement. Can refer to any Java keyword as an example only.
* Use meaningful names
* Make sure that any text appearing within any flowchart symbol provides enough and detailed information for coding this step of the algorithm.
* Finally, the algorithm must be detailed enough for any programming language so any programmer must be able to write the code of the application without asking any question about the algorithm.

An Li, Sagar Patel and Dimitri Sevastis

# Project Pseudocode

Package and class hierarchy

1. main
   1. PhysicsInterface (interface)
   2. ClsStart
   3. ClsButtonsAndFunctions
2. MechanicsBean
   1. Newton2LawClass
   2. ProjMotionClass
3. ENMBean
   1. CoulombClass
   2. OhmClass
4. WavesBean
   1. SHMClass
   2. SingleSlitClass
5. NYA
   1. Newton2ndLaw
   2. ProjectileMotion
6. NYB
   1. CoulombLaw
   2. OhmLaw
7. NYC
   1. SimpleHarmonicMotion
   2. SingleSlitDiffraction

Package 1: main

Interface: PhysicsConstants

* Create final constants for:
  + Coulomb's constant
  + Gravitational Y-acceleration
  + Projectile motion X-acceleration
  + Zero
  + Converting angles in radians to degrees, and degrees to radians.

Class 1: ClsStart extends class Application

* Initialize the program with a protected Border pane that contains a protected menu bar (contained in a protected VBox created with createMenuBar method) with menus "Mechanics", "Electricity and Magnetism", "Waves and Modern Physics" and "Exit" at the top of the screen.
* The remaining area of the window will be split horizontally (protected VBox), with the top half containing the animation (HBox) and the bottom half split vertically containing the controls and text fields (protected VBox that contains a HBox for the buttons and a VBox for the text fields) on the left third and one or two graphs (VBox(es)) on the right of it.
* The menu "Mechanics" contains three menu items named "Newton's Second Law", "Projectile Motion" and "In Construction...".
* The menu "Electricity and Magnetism" contains three menu items named "Coulomb's Law", "Ohm's Law" and "In Construction...".
* The menu "Waves and Modern Physics" contains three menu items named "SHM", "Single-slit Diffraction" and "In Construction...".
* A protected Label object named messageLabel initialized to "Welcome to the Physics Simulator. Please select a category above to begin." will be displayed below the menu bar.
* The setMessage method will be used to change/set the text contained in that object.
* A warning dialog box with text "Topic under construction... Click OK to choose another topic." is displayed if the user clicks the "In construction" item. When the OK button is clicked, user returns to the welcome screen.
* The Exit menu only contains the Exit item. When clicked, a dialog box with text "Thank you for using the Physics Simulator. We hope to see you soon." will be displayed. The OK button of that window exits the program.
* User selects a course to view an animation, one or two graphs, five buttons named Start, Done, Pause, Reset and Help and text fields pertaining to the variables relevant to that course and the messageLabel will display the topic chosen.
* When user changes topics, everything from the previous topic selected will be cleared off and objects pertaining to the new topic will be added.

Class 2: ClsButtonsAndFunctions (abstract)

* Create buttons named Start, Done, Pause, Reset and Help that will be added to the screen.
* Start button will calculate the remaining variable(s), start the animation, and plot the graph(s).
* Done button will return to the main screen.
* Pause button will pause the animation.
* Reset button will clear every text field and graph and also return the animation to the starting state.
* Help button will display a dialog box containing the appropriate help text.
* Create a text object named helpText that will contain the common part of the help text (i.e.: what each button does). The topic-specific part of the help text (i.e.: range of acceptable values, what can/cannot be left blank) will be added in helpText when method showHelpDialogBox is defined from the topic classes.
* Create method doneButtonMethod which will return to the main screen.
* Create method showHelpDialogBox which will show a dialog box with the help text in it.
* Since there's no commonality in the objects used for the other buttons between topic classes, their specific actions will be declared in the topic classes themselves.
* Create abstract methods addTextFields that will add text fields to the screen, addGraphics which will add the animation graphics and graph(s)' background(s), animate that will animate the objects, calculateAllData to calculate data, and graph to create the graphs. All of these methods will be implemented in other classes.

Package 2: MechanicsBean

Class 1: Newton2LawClass

* Define private DoubleProperties for: mass, force, friction force, acceleration, velocity, time and net force.
* Create set and get methods for each defined DoubleProperty, except for net force in which only a get method is required and velocity and time will not have any set or get methods.
* Depending on what field(s) user leaves blank, the following methods are used to do the corresponding calculation:
  + Create method getForceFromFriction with no arguments to compute the friction force (calculated by f = F - Fnet).
  + Create method getFrictionFromForce with no arguments to compute the force (calculated by F = Fnet + f).
  + Create method calculateMass with no arguments to compute the mass (calculated by m = Fnet/a), and set value for corresponding DoubleProperty.
  + Create method calculateNetForce with no arguments to compute the net force (if the user defines both force and friction force) (calculated by Fnet = F - f), and set the value for corresponding DoubleProperty.
  + Create method calculateNetForceWithOneUndefinedForce with no arguments to compute the net force (if the user leave either the force or the friction force undefined) (calculated by Fnet = m × a)
  + Create method calculateAcceleration with no arguments to compute the acceleration (calculated by a = F/m), and set value for corresponding DoubleProperty.
  + Create method graphVelocityAndTime with no arguments which will compute the velocity/time ratio for graphing purposes (v = at).

Class 2: ProjMotionClass implements interface PhysicsConstants

* Define private DoubleProperties for: x displacement, initial height, global x velocity, initial and final y velocities, initial and final velocity magnitude and direction, and time.
* Create set and get methods for each defined DoubleProperty, except for global x velocity, initial and final y velocities and y displacement in which only a get method is required.
* Depending on what field(s) user leaves blank, the following methods are used to do the corresponding calculation:
  + Create method getXVelocity with no arguments to compute the initial X velocity (calculated by vx = vi × cosθ) with the angle converted to radians.
  + Create method getInitialYVelocity with no arguments to compute the initial Y velocity (calculated by viy = vi × sinθ) with the angle converted to radians.
  + Create method getFinalYVelocity with no arguments to compute the final Y velocity (calculated by vfy = vf × sinθ) with the angle converted to radians.
  + Create method getHeightOverTime with no arguments to compute the height according to time (calculated by ∆y = viy∆t + ½ay × ∆t2 + h).
  + Create method calculateInitialVelocityMagnitude with no arguments to compute the initial velocity magnitude (calculated by vi = sqrt(viy2 + vx2)) with the angle converted to degrees.
  + Create method calculateFinalVelocityMagnitude with no arguments to compute the final velocity magnitude (calculated by vf = sqrt(vfy2 + vx2)) with the angle converted to degrees.
  + Create method calculateInitialYVelocity with no arguments to compute the initial Y velocity if final velocity is known (calculated by viy = sqrt(vfy2 - 2 × ay × ∆y)).
  + Create method calculateFinalYVelocity with no arguments to compute the final Y velocity if initial velocity is known (calculated by vfy = sqrt(viy2 + 2 × ay × ∆y)).
  + Create method calculateTime with no arguments to compute the time (calculated by ∆t = ∆x/vx).
  + Create method calculateVelocityMagnitudeOverTime with no arguments to compute the time (calculated by v = sqrt(vx2 + (viy + ay∆t)2)).
  + Create method calculateXDisplacement with no arguments to compute the x displacement (calculated by ∆x = ∆t × vx).
  + Create method calculateXVelocity with no arguments to compute the X velocity given the time and displacement (calculated by vx = ∆x/∆t).
  + Create method calculateInitialVelocityDirection with no arguments to compute the initial pitch angle (calculated by θi = tan-1(viy/vx)) then converted to degrees.
  + Create method calculateFinalVelocityDirection with no arguments to compute the final landing angle (calculated by θf = tan-1(vfy/vx)) then converted to degrees.

Package 3: ENMBean

Class 1: CoulombClass implements interface PhysicsConstants

* Define private DoubleProperties for: Source charge, test charge, distance of separation between the objects, and electrical force.
* Create set and get methods for each defined DoubleProperty.
  + Create method calculateEForce with no arguments to compute the magnitude of the electric force between the two charged objects (calculated by F = k × abs(q1×q2)/(r2)), and set this value for corresponding DoubleProperty.
  + Create method calculateDistance with no arguments to computer the distance of seperation between the two charged objects (calculated by r = sqrt(k × abs(q1×q2)/F)), and set this value for corresponding DoubleProperty.

Class 2: OhmClass

* Define private DoubleProperties for: voltage, current, and resistance.
* Create set and get methods for each defined DoubleProperty.
* Depending on what field user leaves blank, the following methods are used to do the corresponding calculation:
  + Create method calculateVoltage with no arguments to compute the voltage (calculated by V = I × R), and set value for corresponding DoubleProperty.
  + Create method calculateResistance with no arguments to compute the voltage (calculated by R = V/I), and set value for corresponding DoubleProperty.
  + Create method calculateIntensity with no arguments to compute the voltage (calculated by I = V/R), and set value for corresponding DoubleProperty.

Package 4: WavesBean

Class 1: SHMClass

* Define private DoubleProperties for: amplitude, time, phase constant, frequency, period, angular velocity, linear velocity and Y-displacement.
* Create set and get methods for each defined DoubleProperty.
  + Create method calculateAngularVelocity with no arguments to compute the angular velocity (calculated by ω = 2π × f), and set value for corresponding DoubleProperty.
  + Create method calculatePeriod with no arguments to compute the period (calculated by T = 1/f), and set value for corresponding DoubleProperty.
  + Create method calculateFrequency with no arguments to compute the frequency (calculated by f = 1/T) and set value for corresponding DoubleProperty.
  + Create method calculateYDisplacement with no arguments to compute the Y-displacement (y(t) = Asin(ωt) + φ0)
  + Create method calculateLinearVelocity with no arguments to compute the linear velocity (v(t) = ωAcos(ωt) + φ0)

Class 2: SingleSlitClass implements interface PhysicsConstants

* Define private DoubleProperties for: slit width, angle, wavelength, distance and Y-displacement.
* Define a private IntegerProperty for the diffraction order.
* Create set and get methods for each defined Property.
* Depending on what field(s) user leaves blank, the following methods are used to do the corresponding calculation:
  + Create method calculateSlitWidth with no arguments to compute the slit width (calculated by a = mλD/y).
  + Create method calculateAngle with no arguments to compute the angle (calculated by θ = tan-1(y/D)) then converted to degrees.
  + Create method calculateWavelength with no arguments to compute the wavelength (calculated by λ = ay/mD) then converted to degrees.
  + Create method calculateDistance with no arguments to compute the distance between the slit and the screen (calculated by D = ay/mλ).
  + Create method calculateYDisplacement with no arguments to compute the Y-displacement (calculated by y = mλD/a).
  + Create method calculateOrder with no arguments to compute the diffraction order, rounded to the nearest integer (calculated by m = λD/ay).

Package 5: NYA

Class 1: Newton2ndLaw extends ClsButtonsAndFunctions

* Create protected text fields for force, friction force, mass and acceleration.
* Create a block in the center of the screen with two force arrows, one representing force and the other one representing friction (all objects protected).
* Create animations described below for each object defined.
* Import one protected graph image which will be velocity (-20–20 m/s) over time (0–10 seconds) in which the slope of that graph would be equal to the acceleration.
* Define abstract method addTextFields which will add text fields previously defined in the textFieldBox.
* Define abstract method addGraphics which will add the objects being animated in the animationBox and the graph background in the graphBox1.
* Define abstract method calculateAllData that uses the set methods in class Newton2LawClass to set the corresponding inputs to the properties in that class and depending on what field the user leaves blank, use the appropriate method(s) for calculating the remaining variable:
  + If the user leaves force blank, use methods calculateNetForceWithOneUndefinedForce and calculateForceFromFriction to calculate it.
  + If the user leaves friction force blank, use methods calculateNetForceWithOneUndefinedForce and calculateFrictionFromForce to calculate it.
  + If the user leaves mass blank, use methods calculateNetForce and calculateMass to calculate it.
  + If the user leaves acceleration blank, use methods calculateNetForce and calculateAcceleration to calculate it.
* Show a warning dialog box if mass is negative with an OK button that will return to the previous screen and a Help button that will display the help dialog.
* Define abstract method animate in which the block and arrows will grow/shrink depending on the mass and force values.
* The block will translate and accelerate depending on the acceleration direction (right for positive or left for negative).
* The force arrow will point right if force is positive or left if force is negative. The friction force arrow will do the opposite thing.
* Define abstract method graph to create a line chart using velocity and time.

Class 2: ProjectileMotion extends ClsButtonsAndFunctions and implements interface PhysicsConstants

* Create protected text fields for initial and final velocity magnitude and direction, total time, height and total X-displacement.
* Create a rising platform on the left of the screen and a small circle object on it (all objects protected).
* Create animations described below for each object defined.
* Import two protected graph images which will be height (0–10 m) over time (0-10 s) and velocity magnitude (0-50 m/s) over time (0-10 s).
* Define abstract method addTextFields which will add text fields previously created in the textFieldBox.
* Define abstract method addGraphics which will add the objects being animated in the animationBox and the graphs backgrounds in the graphBox1 and graphBox2.
* Define abstract method calculateAllData which will use the set methods in class ProjMotionClass to set the corresponding inputs to the properties in that class and methods getXVelocity, getInitialYVelocity and getFinalYVelocity to break up initial and final velocity magnitude and direction into x and y components and depending on what field the user leaves blank, use the appropriate method(s) for calculating the remaining variable(s).
  + If user leaves the initial velocity magnitude blank, use methods calculateXVelocity, calculateInitialYVelocity and calculateInitialVelocityMagnitude to compute it.
  + If user leaves the final velocity magnitude blank, use methods calculateXVelocity, calculateFinalYVelocity and calculateFinalVelocityMagnitude to compute it.
  + If user leaves the initial velocity direction blank, use method calculateInitialVelocityDirection to compute it.
  + If user leaves the final velocity direction blank, use method calculateFinalVelocityDirection to compute it.
  + If user leaves both initial velocity magnitude and direction blank, use methods calculateInitialVelocityMagnitude and calculateInitialVelocityDirection to compute it.
  + If user leaves both final velocity magnitude and direction blank, use methods calculateInitialVelocityMagnitude and calculateInitialVelocityDirection to compute it.
  + If user leaves time blank, use method calculateTime to compute it.
  + If user leaves total X-displacement blank, use method calculateXDisplacement to compute it.
* Show a warning dialog box if user leaves height blank, if any value is negative or out of the graphing range with an OK button that will return to the previous screen and a Help button that will display the help dialog.
* Define abstract method animate that will translate the circle object in a conventional projectile trajectory.
* Define abstract method graph to create line charts using height and time for the first graph and velocity magnitude over time for the second.

Package 6: NYB

Class 1: CoulombLaw extends ClsButtonsAndFunctions and implements PhysicsInterface

* Create protected text fields for source charge, test charge, and distance between the two charged objects.
* Create two protected circle objects to represent each charged object, with a fill color of red or blue depending on where the charge is positive or negative respectively.
* Create animations described below for each object defined.
* Import one protected graph image which will be force×distance2 0–106 Nm2 versus the absolute value of the product of the two charges (0 μC–100 μC) in which the slope of that graph would be equal to k.
* Define abstract method addTextFields which will add text fields previously created in textFieldBox.
* Define abstract method addGraphics which will add the objects being animated in the animationBox and the graph background in the graphBox1.
* Define abstract method calculateData that uses the set methods in class CoulombClass to set the corresponding inputs to the properties in that class and depending on what field the user leaves blank, use the appropriate method(s) for calculating the remaining variable:
  + If the user leaves electric force blank, use method calculateEForce to calculate it.
  + If the user leaves distance blank, use method calculateDistance to calculate it.
* Show a warning dialog box if user leaves source charge, test charge or both blank with an OK button that will return to the previous screen and a Help button that will display the help dialog.
* Define abstract method animate in which the two circles will horizontally translate to adequately display the inputted/calculated distance.
* Depending on inputted/calculated electric force, as well as whether the object charges are same/opposite, a force vector (in the form of an arrow) will be animated and:
  + Its length will depend on the electric force magnitude
  + Its direction will depend on the charges.
* Define method graph to create a line chart using the appropriate data.

Class 2: OhmLaw extends ClsButtonsAndFunctions

* Create protected text fields for voltage, intensity, and resistance.
* Create protected rectangle with black border and no fill to represent circuit.
* Import protected images of battery and resistor that will go along circuit.
* Import protected image of arrow (with right angle) that will go along circuit border.
* Create animations described below for each object defined.
* Import protected graph image which will be voltage (0–10 V) versus resistance (0–1000 Ω) in which the slope of the graph will be equal to intensity.
* Define abstract method addTextFields which will add text fields previously created in textFieldBox.
* Define abstract method addGraphics which will add the objects being animated in the animationBox and the graph background in the graphBox1.
* Define abstract method calculateData that uses the set methods in class OhmClass to set the corresponding inputs to the properties in that class and depending on what field the user leaves blank, use the appropriate method(s) for calculating the remaining variable:
  + If the user leaves voltage blank, use method calculateVoltage to calculate it.
  + If the user leaves intensity blank, use method calculateIntensity to calculate it.
  + If the user leaves resistance blank, use method calculateResistance to calculate it.
* Show a warning dialog box if any variable has a negative value or out of the graphing range with an OK button that will return to the previous screen and a Help button that will display the help dialog.
* Define abstract method animate in which:
  + Depending on voltage (>1.0V) the battery image will scale in size.
  + Depending on resistance (>10 Ω) the resistor image will scale in size.
  + Depending on intensity (>0.1 mA) the arrow image will scale in size.
* Define method graph to create a line chart using the appropriate data.

Package 7: NYC

Class 1: SimpleHarmonicMotion extends ClsButtonsAndFuctions

* Create protected text fields for amplitude, phase constant, frequency, period and angular velocity.
* Create a square to represent the block in SHM, a vertical line to represent the wall and an image representing the spring (all objects protected).
* Create animations described below for each object defined.
* Import two protected graph images which will be y displacement over time and velocity over time.
* Define abstract method addTextFields that will add text fields previously created in the textFieldBox.
* Define abstract method addGraphics which will add the objects being animated in the animationBox and the graph backgrounds in the graphBox1 and graphBox2 respectively.
* Define abstract method calculateAllData that uses the set methods in class SHMClass to set the corresponding inputs to the properties in that class.
* Depending on what field the user leaves blank, use the appropriate method(s) for calculating the remaining variable:
  + If the user leaves angular velocity blank, use method calculateAngularVelocity to calculate it.
  + If the user leaves period blank, use method calculatePeriod to calculate it.
  + If the user leaves frequency blank, use method calculateFrequency to calculate it.
  + If the user leaves Y-displacement blank, user method calculateYDisplacement to calculate it.
  + If user leaves phase constant blank, store 0 in that property.
* Show a warning dialog box if frequency, period or amplitude is negative or if any value is out of the graphing range with an OK button that will return to the previous screen and a Help button that will display the help dialog.
* Show a dialog box with text “Phase constant is left blank. Program will assume that it is 0. Continue?” if user leaves phase constant blank. User will click OK to continue with that value or Cancel to change it.
* Define abstract method animate that will translate the square object in a conventional SHM trajectory.
* Define abstract method graph to create line charts using y displacement and time for the first graph and velocity and time for the second.

Class 2: SingleSlitDiffraction extends ClsButtonsAndFuctions and implements PhysicsConstants

* Create protected text fields for slit width, angle, wavelength, distance, diffraction order and Y-displacement.
* Create 2 horizontal lines (one representing the distance between the slits and the wall and another representing the laser slightly), three vertical lines (2 representing the top and bottom of the slit and the other representing the wall), one dotted line connecting the middle of the slit to the top of the first diffraction on the wall and one image representing the diffraction pattern (all objects protected).
* Create animations described below for each object defined.
* Import one protected graph image which will be y displacement (0–1 cm) over angle (0–45 degrees).
* Define abstract method addTextFields that will add text fields previously created in the textFieldBox.
* Define abstract method addGraphics which will add the objects being animated in the animationBox and the graph background in the graphBox1.
* Define abstract method calculateAllData that uses the set methods in class SingleSlitClass to set the corresponding inputs to the properties in that class.
* Depending on what field the user leaves blank, use the appropriate method(s) for calculating the remaining variable:
  + If the user leaves slit width blank, use method calculateSlitWidth to calculate it.
  + If the user leaves angle blank, use method calculateAngle to calculate it.
  + If the user leaves wavelength blank, use method calculateWavelength to calculate it.
  + If the user leaves distance blank, use method calculateDistance to calculate it.
  + If the user leaves Y-displacement blank, use method calculateYDisplacement to calculate it.
  + If the user leaves diffraction order blank, use method calculateOrder to compute it.
* Show a warning dialog box if any variable has a negative value or out of the graphing range with an OK button that will return to the previous screen and a Help button that will display the help dialog.
* Define abstract method animate that will scale vertically the two slit lines according to the slit width, scale the diffraction pattern according to the y displacement and the angle, and translate the endpoint of the dotted line to the first diffraction top point.
* Define abstract method graph to create line charts using y displacement and angle for the graph.