CSC1130 Introduction to Computing Using Java 2019-2020 First Term
Department of Computer Science and Engineering
The Chinese University of Hong Kong

Due date: 16 December 2019 (Mon) Mini Project Full mark: 100
Expected normal time spent: 9 hours

Chart

Aim: 1. build a practical GUI Chart application using Java.
2. practise using existing classes and objects.
3. practise sub-classing and implement interface.

Task: Create a Java application for charting!

Background on Sub-Classing and Interface:

1. Kick start our last lecture on sub-classing and interface with a busy scenario:

   a) *a computer* is a machine that can store information, perform calculations, and make logical decisions.
   b) *a tablet* is a machine that can store information, perform calculations, make logical decisions, support a large pen-input screen, and run on battery (thus portable.)
   c) *a smartphone* is a machine that can store information, perform calculations, make logical decisions, support a medium-size screen, make phone call, and run on battery (thus portable.)
   d) *a mobile* device is a device that can run on battery (thus portable.)

   Imagine that we want to model the entities listed above in OOP but we find that many descriptions (fields and methods) are similar or even identical!

   If Computer is a class. Computer bears fields and methods.

   We want to define another class Tablet. Shall we start from scratch OR copy code from existing class Computer? If we copy code, what if there are future modifications to the class Computer? Shall we copy again and again? We need a new tool: Sub-Classing (**Inheritance**.)

   We say, a tablet is a computer that can support a large screen, and run on battery.
   So class Tablet is a *sub-class* of Computer, with certain extensions.
   In Java, we write

```
class Tablet extends Computer
{
    // Java understands that all fields and methods are inherited
    // from Computer in the subclass Tablet.  We need not copy code.
    // There are also new features:
    public void configurePenScreen() {...}

    public void runOnBattery() {...}
}
```

   Similarly, `class SmartPhone extends Computer {...}`

   Class Computer is a *super-class* or *base-class*.

Notice that there are many kinds of portable devices apart from tablets and smartphones. For example, analog watch, torch, portable electric fan, etc. They all run on battery but they are fundamentally different. We shall neither define something like SmartPhone extends ElectricFan, nor AnalogWatch extends Tablet. How to model the idea of "RunOnBattery" then?

Java provides an interface mechanism:

```
interface Portable {
    public void runOnBattery(); // no code here!
}

class Tablet extends Computer implements Portable {...}

class SmartPhone extends Computer implements Portable {...}

class AnalogWatch extends Watch implements Portable {...}

class Torch implements Portable {...}

class PortableElectricFan implements Portable {...}
```

Each and every "Portable"-compliant class shall implement the **Portable** interface by defining its own **runOnBattery()** method. All such objects are instances of type **Portable**.

Comprehensive example:

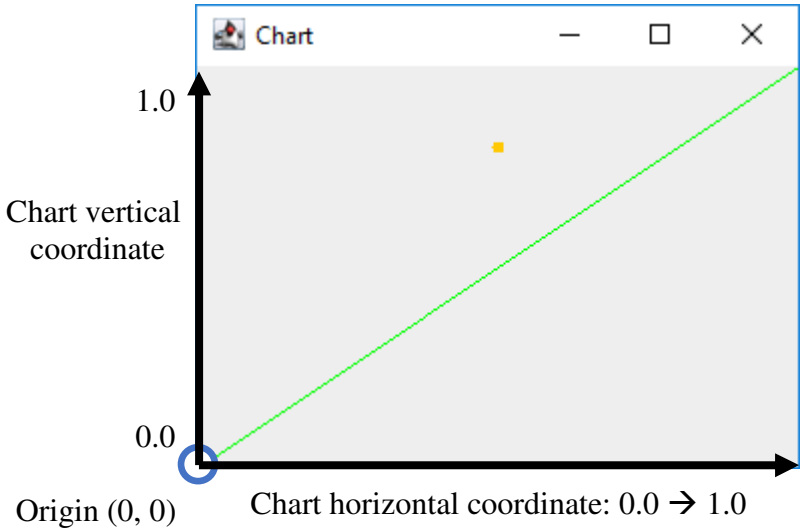
```
SmartPhone phone = new SmartPhone();
Tablet      tab1 = new Tablet();
Computer    tab2 = new Tablet(); // Tablet is a subclass of Computer
AnalogWatch   w1 = new AnalogWatch();
Watch   w2 = new AnalogWatch(); // AnalogWatch is a subclass of Watch

Portable device;
device = phone; // copy reference; SmartPhone implements Portable
device.runOnBattery();
device = tab1; // copy reference; Tablet implements Portable
device.runOnBattery();
device = w1; // copy reference; AnalogWatch implements Portable
device.runOnBattery();
if (tab2 instanceof Portable)
{
    System.out.println("Object var tab2 is referencing to a Tablet
object which is compatible to (instanceof) type Portable.");
    device = (Portable) tab2;  // type casting to Portable
    device.runOnBattery();
}
```

Tablet is a sub-type of Computer.
SmartPhone is a sub-type of Computer.
AnalogWatch is a sub-type of Watch.
Tablet, SmartPhone, AnalogWatch, Torch, PortableElectricFan are all Portable.

2. A project skeleton is provided. A super-class Chart is given. Chart handles most of the windowing and GUI stuff. We are going to define new sub-classes to inherit and enjoy those Chart capabilities, as well as to provide further extensions.

3. To let you understand the basic operations, there is already a given main() method in class Chart. Feel free having some fun. The Chart coordinates system is simple and conventional, origin is at bottom-left corner, x-axis grows right while y-axis *grows up*, x-y coordinates of the shapes (dots and lines) go from 0.0 to 1.0 irrespective of screen and window dimensions.



4. We want to have two sub-classes of Chart, FunctionChart and ScatterChart.

5. FunctionChart is given as an example for your reference. FunctionChart objects can plot the graph of a given Function object y = f(x) with certain ranges of x and y.

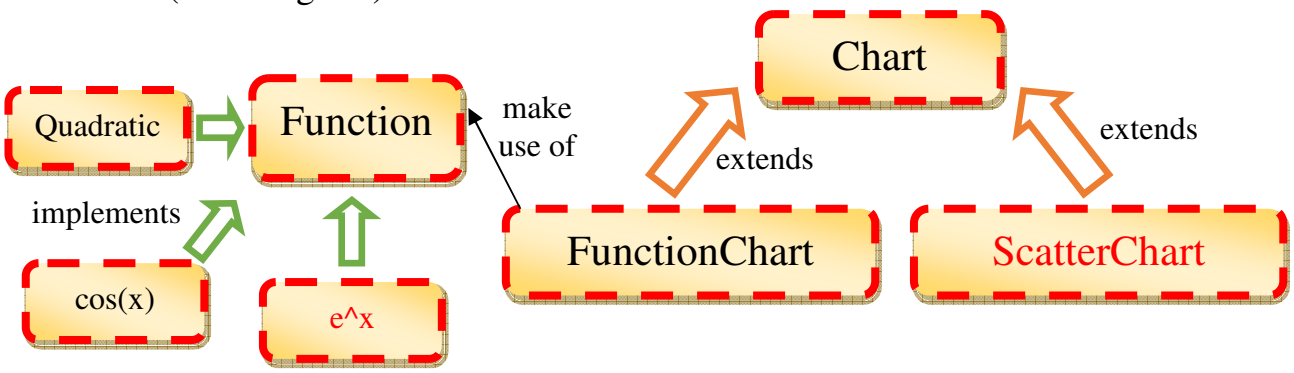
   Interestingly, Function is an interface. We are free to define many classes that implement the Function interface. The key interface method is valueAt(x). We have provided you two sample classes QuadraticFunction Ax^2+Bx+C and CosineFunction cos(x). You are expected to define the ExponentialFunction e^x.

6. You shall complete another sub-class, ScatterChart. It accepts two arrays of X[ ] and Y[ ] values and plots the X-Y pairs. Mind you that you shall find out and determine the ranges of the values in order to scale your scatter plot and position your X-Y axes. Dot size is fixed and inherited from class Chart and dots should be in ORANGE. X-Y axes are in BLUE.

7. In summary, you have to work on and complete TWO classes:
   ExponentialFunction which implements Function; and
   ScatterChart which extends Chart. ScatterChart has one constructor that takes two arrays.
   There are plenty of reference code in the provided classes for your reference and experiment.

   Note: window resizing is NOT supported and you need NOT handle it.
   Remark: our sample ScatterChart frames the axes and dots with an additional boundary of 10 pixels around the plot, to make it better looking. This feature is optional.

Overview (OO Diagram):

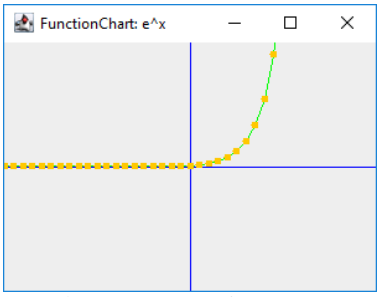


Procedure (Step-by-Step):

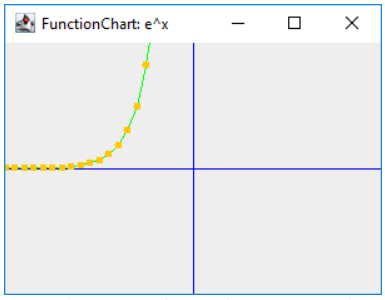
1. Download given **`ChartGivenProject.zip`** from Blackboard.

2. Open the given NetBeans Java Application project **`Chart`**. Apply suggested package name **`chart`** and study the main() method in class **`Main`** and the main() method in class **`Chart`**.

3. Study reference code given in CosineFunction, finish your own ExponentialFunction.

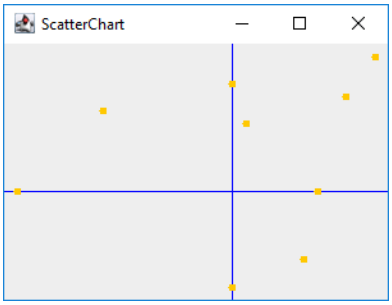


Sample: e^x macro view

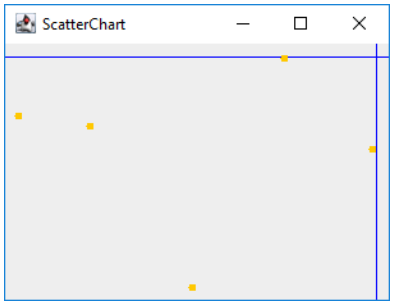


Sample: e^x micro view (zoom-in near zero)

4. Study reference code given in FunctionChart, finish your own ScatterChart.



Sample: scatter1 (X1[ ], Y1[ ])



Sample: scatter2 (X2[ ], Y2[ ])

5. Test well and debug.

6. ZIP the whole NetBeans project folder **`Chart`** and Submit the file **`Chart.zip`** via our online Assignment Collection Box on Blackboard.

Marking Scheme and Notes:

1. The submitted program should be free of any typing mistakes, compilation errors and warnings.

2. Comment/remark, indentation, style are under assessment in every programming assignments unless specified otherwise. Variable naming, proper indentation for code blocks and adequate comments are important. Include also your personal particulars and your academic honesty declaration statement in a header comment block of each source file.

3. Remember to upload your submission and **click Submit** before 18:00 p.m. of the due date. No late submission would be accepted.

7. If you submit multiple times, ONLY the content and time-stamp of the latest one would be counted. You may delete (i.e. take back) your attached file and re-submit. We ONLY take into account the last submission.

University Guideline for Plagiarism:

Attention is drawn to University policy and regulations on honesty in academic work, and to the disciplinary guidelines and procedures applicable to breaches of such policy and regulations. Details may be found at http://www.cuhk.edu.hk/policy/academichonesty/.

With each assignment, students are required to submit a statement that they are aware of these policies, regulations, guidelines and procedures, in a header comment block.

Faculty of Engineering Guidelines to Academic Honesty:
MUST read: https://www.erg.cuhk.edu.hk/erg/AcademicHonesty
(you may need to access via CUHK campus network/ CUHK1x/ CUHK VPN)

Further Possibilities (ungraded optional extensions):

1. Can you add text to your charts?

2. Try adding a new type BarChart.