CS52 Lab8

**Inheritance from Abstract Class(es)**

**Note:**

Please understand that you are not allowed to alter any of the followings from what is given in this design document. I will explain the reasons for that later, but it is a standard software engineering practice that once design decides the names of followings, the programmer is not free to alter them without having the meeting with all stakeholders again and reaching a new decision on new names. **The un-alterable names are:**

1. **Class names**
2. **Class member names**
3. **Names of files that store source code of your classes**
4. **Class member data types, return types, and types and order of arguments passed to member functions and constructor(s).**

Programmer does have complete freedom to choose names of function arguments, local variables, and algorithms.

**Each source code file must have your full ID block as comments on top of file that must include followings:**

*Student Name*

*Project Name*

*Start Date*

*Completion Date*

*Operating System Used*

*Compiler Used*

*Student Email*

**About Specifications:**

Best attempt has been made to present bulk of specifications in this document. However, additional specifications are contained not just in the descriptions, but also in documented output examples, general software industry practices and verbal instructions. Your task includes building a clear vision of specifications and coding, and testing your software to them. Please read this whole document before jumping to code. Set aside distraction free 3/4 hour to read this document and write down questions on it! [That means that you should work off a paper copy].

**Background**

In enterprise level software products, code re-use, and code maintainability are the key factors to keep the original software cost, and maintenance costs low. Both these objectives are in part achievable by using a technology called inheritance. Code re-use allows building a new software component by using parts of the existing code. On the other hand, OOP in general and inheritance in particular allows fault detecting the modules in software that need maintenance, thus reducing maintenance cost. In this project we are using inheritance to derive a Manager class from an abstract class Employee. Abstract classes are a powerful way to express pure design as they include pure virtual functions, which can be implemented in fully customizable ways by the inherited classes. We will lay out the specifications for the Abstract class, Employee, derived concrete class Manager, and the goals to be accomplished by test code and main function. UML diagram for both classes is given on next page, including their inheritance hierarchy. [Caution: UML diagrams are NOT C++ source codes!].



**Specifications of Employee Class**

Specifications are written in terms of description of class members. Please feel free for further explanation of class member and their purpose. You are responsible for developing algorithm, coding and testing each member function.

### Detailed Description of class Employee

The class **Employee** is an abstract class that contains data members:

* FirstName
* LastName

**Pure Virtual** member functions as below:

* getFirstName
* getLastName
* getFullName
* setLastName
* setFirstName
* toString

Class also has one default/explicit constructor that does double duty as both; default and explicit constructors.

The header file in which **Employee** class will be placed will be called **Employee.h**.

The below gives an example of a pure virtual function.

**class Employee**

**{**

**protected:**

**//protected data members are declared here.**

**public:**

**virtual const string getFirstName() const = 0; //Example of declaration of pure virtual function.**

**};**

Understand the difference between virtual function and pure virtual function.

### Member Data Documentation For Class Employee

#### string FirstName[protected]// protected data members are under word protected:

The instance field FirstName holds/stores the first name of employee.

#### string LastName[protected]

The instance field LastName holds/stores the last name of employee.

#### const size\_t MIN\_NUM\_CHARS\_NAMES = 3[static], [protected]

**static** const int storing, minimum number of characters required in either the first name, last name, and later in Manager class, bosses title.

### Summary Only Of Public Member Functions

Employee (const string & Fname="First name not set", const string & Lname="Last Name not set")

Virtual const string getFirstName () const =0

virtual const string getLastName () const =0

virtual const string getFullName () const =0

virtual const string toString () const =0

virtual void setFirstName (const string & Fname)=0

virtual void setLastName (const string & Lname)=0

### Summary only of Protected Attributes or Data Members

string FirstName

string LastName

### Summary only of Static Protected Attributes

static const int MIN\_NUM\_CHARS\_NAMES = 2

### Detailed Documentation of Employee Class Member functions and Constructor

### Constructor Documentation

#### Employee (const string & *Fname* = "First  name  not  set", const string & *Lname* = "Last  Name  not  set")

Employee class is abstract and cannot be instantiated. But it still must have a constructor, that is used to construct data members FistName, and LastName, that are inherited by it’s child classes. Default constructor would initialize first and last name fields to:

1. Both first name and last name are set to default values of either "First Name not set", or "Last name not set", respectively.

**Explicit constructor scenarios are discussed below:**

1. First name can be set to (user defined), explicit value and last name to default value of "Last name not set".
2. Last name can be set to (user defined), explicit value, and first name to default value, “First name not set”.

4. Both first and last names can be set to (user defined) explicit values. Constructor does data validation when explicit values for first name only, or for both first and last names are provided. The validation algorithm ascertains that explicit values of first and last names contain at least two characters. If explicit values are not meeting this requirement, then first or last names or both are set to default values. You are responsible for designing, coding, and testing the algorithm for the constructor. Please ask question if you are not clear about the goals of the constructor.

##### Parameters:

|  |  |
| --- | --- |
| *Fname* | is the first name of **Employee** |
| *Lname* | is the last name of **Employee** |

### Member Function Documentation [Given in alphabetical order. This is not the coding and testing order!]

**Note: Pure virtual functions have no code ever!**

#### virtual const string getFirstName () const[pure virtual]

Pure virtual function getFirstName, **when implemented** by the concrete derived class will implement full body of getFirstName, in which case it would return the first name of **Employee**. Non-implementation by the derived class will make the derived class an abstract class.

##### Returns:

the first name of employee when implementation is complete.

#### virtual const string getFullName () const[pure virtual]

Pure virtual function getFullName, **when implemented** by the concrete derived class will implement full body of getFullName, in which case it would return the full name of **Employee**. Non-implementation by the derived class will make the derived class an abstract class.

##### Returns:

the full name of employee when implementaion is complete.

#### virtual const string getLastName () const[pure virtual]

Pure virtual function getLastName, **when implemented** by the concrete derived class will implement full body of getLastName, in which case it would return the last name of **Employee**. Non-implementation by the derived class will make the derived class an abstract class.

##### Returns:

the last name of employee when implementaion is complete.

#### virtual void setFirstName (const string & *Fname*)[pure virtual]

Pure virtual function setFirstName **when implemented** by the deriving class sets the first name of **Employee** to new mutated value while still enforcing the invariant that new first name is not less than 2 characters. Non-implementation by the derived class will make the derived class an abstract class.

##### Parameters:

|  |  |
| --- | --- |
| *Fname* | is new first name of **Employee**. |

#### virtual void setLastName (const string & *Lname*)[pure virtual]

Pure virtual function setLastName **when implemented** by the deriving class sets the last name of **Employee** to new mutated value while still enforcing the invariant that new last name is not less than 2 characters. Non-implementation by the derived class will make the derived class an abstract class.

##### Parameters:

|  |  |
| --- | --- |
| *Lname* | is new last name of **Employee**. |

#### virtual const string toString () const[pure virtual]

Pure virtual function toString **when implemented** by the deriving class returns a string that has information about all class data members. Typical returned string may look like below:

**Employee** Name: John Doe

**Employee** Title: **Manager**

**Employee** Salary: $2000.00 per month

Non-implementation by the derived class will make the derived class an abstract class.

##### Returns:

string that contains **Employee** data details.

There is no testing needed of abstract class Employee as one cannot create an instance of an abstract class. Abstract classes exist only to serve as a base class or to be derived from. Testing will be done for next class Manager that derives from abstract class Employee.

**Description of Derived Class (from Employee) Manager**

### Detailed Description (Please see UML diagram also for pictorial representation)

Class **Manager** extends or derives from abstract class **Employee**. Apart from inheriting base class data members FirstName and LastName, the class **Manager** adds following of its own protected data members:

* Salary
* TitleOfBoss

Class **Manager** implements all pure virtual functions from its base class **Employee** and has get and set functions for additional data members that it introduced. Class also defines its own constructor and **calls base class constructor** to build the base class portion of **Manager** object.

**Note:** **The syntax of calling base class constructor has been illustrated in the inheritance example for Vehicle base class done in the lecture. Copy of that code is available on team website as well as at the e-book website [sites.google.com/site/cplusplussmc].**

### Summary of Public Member Functions of Manager Class

Manager (const string & Fname="First Name not set", const string & Lname="Last name not set", double sal=0.0, const string & BTitle="Boss'es Title not set")

const string getFirstName () const

const string getLastName () const

const string getFullName () const

const string toString () const

void setFirstName (const string & Fname)

void setLastName (const string & Lname)

double getSalary () const

const string getBossTitle () const

void setSalary (double sal)

void setBossTitle (const string & Title)

### Summary of Protected Attributes or Data Members of Manager class

double salary

string TitleOfBoss

Constructor Documentation

**Caution: Initialization list MUST be used in implementing all constructors. Points would be taken off for not using initialization list. We have shown implementation of constructor initialization list in tremendous depth in code examples done in the class. Study those examples.**

#### Manager (string *Fname* = "First  Name  not  set", string *Lname* = "Last  name  not  set", double *sal* = 0.0, string *BTitle* = "Boss'es  Title  not  set")

Default and explicit constructor for **Manager** class. Constructor would still need the arguments for initializing the inherited data members.

##### Parameters:

|  |  |
| --- | --- |
| *Fname* | is the first name of manager. |
| *Lname* | is the last name of manager |
| *sal* | is the salary of **Manager** |
| *BTitle* | is the title of **Manager**'s boss |

### Details of Member Function Documentation of Manager Class

The members are given in alphabetical order. You should code constructor and toString member function and test both of them. After that **Please ONLY code, and test one member function at a time or the first group that is needed to begin first set of tests. Later we give you space to write your algorithms on a paper copy for each class member function/constructor as needed. Continue this process until you have finished the below cycle for each member function:**

1. **Write algorithm for the member function**
2. **Code the algorithm for the member function.**
3. **Test the member function that algorithm of step one is coded correctly.**

**I may ask to see your algorithms and test code and its output!**

#### Const string getBossTitle () const [virtual]

Returns the title of **Manager**'s boss

##### Returns:

the title of **Manager**'s boss.

#### const string getFirstName () const[virtual]

Returns the first name of **Manager**

##### Returns:

the first name of the **Manager**

#### const string getFullName () const[virtual]

Returns the full name of **Manager** by concatenating the first and last names.

##### Returns:

the full name of the **Manager**

#### const string getLastName () const[virtual]

Returns the last name of **Manager**

##### Returns:

the last name of the **Manager**

#### double getSalary () const [virtual]

Returns the salary of **Manager**.

##### Returns:

the salary of **Manager**.

#### void setBossTitle (const string & *Title*) [virtual]

setBossTitle functions sets the new title for **Manager**'s boss.

##### Parameters:

|  |  |
| --- | --- |
| *Title* | is the new title for **Manager**'s Boss |

#### void setFirstName (const string & *Fname*)[virtual]

Inherited function setFirstName is implemented by the **Manager** class and sets the first name of **Manager** to new mutated value while still enforcing the invariant that new first name is not less than 2 characters.

##### Parameters:

|  |  |
| --- | --- |
| *Fname* | is new first name of **Manager**. |

#### void setLastName (const string & *Lname*)[virtual]

Inherited Function setLastName is implemented by the **Manager** class and sets the last name of **Manager** to new mutated value while still enforcing the invariant that new last name is not less than 2 characters. If new value violates the invariant then old last name is not changed.

##### Parameters:

|  |  |
| --- | --- |
| *Lname* | is new last name of **Manager**. |

#### void setSalary (double *sal*) [virtual]

setSalary function sets the new salary for the existing **Manager** object.

##### Parameters:

|  |  |
| --- | --- |
| *sal* | is the new salary of **Manager**. It cannot be negative. |

#### const string toString () const[virtual]

toString is implemented by the **Manager** class, and returns a string that has information about all **Manager** data members. Typical returned string may look like below:

***Manager*** *Name: John Doe*

*Boss'es Title: Director*

***Employee*** *Salary: $2000.00 per month*

##### Returns:

string that contains **Manager**'s data details.

**Coding Sequence**

Before beginning to test any piece of code you need to code below components, correctly and debug any errors at that point before proceeding further.

1. Data Members and constructor of Employee class. Though Employee class is an abstract class and cannot be instantiated, still it requires a constructor header (in Employee.h file) and constructor body (in Employee.cpp file), so that later that constructor is usable to construct the Employee part of classes that are derived from Employee. See specifications of Employee class constructor on page 4. Write down **below** the pseudo code (algorithm) for Employee class constructor. [If your constructor does not work, then I may ask you to show me your algorithm for Employee class constructor].
2. Only toString pure virtual function for Employee class. Please see its description on page 5. It would just be one line of code in Employee.h file and no code in Employee.cpp file. [Do not write other pure virtual functions names in Employee class right now because it would make it impossible to test Manager class. That is because instantiation of Manager class would require that it should override all pure virtual functions of Employee class]. You need to add one pure virtual function in Employee class and then code and test it in Manager class after that.
3. Data members and constructor of Manager class. Their descriptions are given on page 6 and 7 of this document. By hand, write in the space below the pseudo code/algorithm for the Manager class constructor.
4. Write the toString function for Manager class.

By now you have written followings:

1. Constructors of Employee class.

2. Constructor for Manager class

3.toString function as pure virtual function for Employee but it’s full implementation in Manager class. The test code in left column of table below must give the results shown in right column. In table below portions of outputs come from toString function and others from two constructors that you wrote. Invariant controlling number of characters in first and last names is enforced in Employee class constructor. The invariant controlling number of characters in Boss’s title is enforced in constructor for Manager class.

|  |  |
| --- | --- |
| Test source code [Do not copy and paste source code to Visual Studio] | Output from Test source code |
| ManagerM1**;**  cout **<<** M1**.**toString**() <<** endl**;**  ManagerM2**("F","L",-200.0, "T");**  cout **<<** M2**.**toString**() <<** endl**;**  ManagerM3**("Fran", "Lamar", 4000.99, "Vice President");**  cout **<<** M3**.**toString**() <<** endl**;** | -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: First Name not set Last name not set  Boss'es Title: Boss'es Title not set  Salary: $0.00  ----------------------------------------------  The first name F does not have enough characters.  We will set first name to a default value of "First name not set."  The last name L does not have enough characters.  We will set last name to a default value of "Last name not set."  Title of boss cannot be less than 2 characters. The default value is used.  -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: First name not set Last name not set  Boss'es Title: Boss'es Title not set  Salary: $200.00  ----------------------------------------------  -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Fran Lamar  Boss'es Title: Vice President  Salary: $4000.99  ---------------------------------------------- |

If at this point the above test shows problems then see TA or Instructor.

1. **Coding the mutator setLastName for both classes.** In Employee class header file you just need one line of code for pure virtual function setLastName. No code for this function is needed in Employee.cpp file. Then you need a corresponding pure virtual function overriding code in header and implementation files of Manager class. Notice that setLastName function in Manager class should also enforce the invariant that last name cannot be less than three characters. If that is the case then previous name (constructed by constructor) is NOT changed. See details of setLastName on pages 5 and 8. In the box below, write the algorithm for setLastName function for Manager class.

Table below shows the source code and output to test the setLastName function.

|  |  |
| --- | --- |
| Source code to test setLastName function .[Do not copy and paste this source code into your program]. | Output from source code in left column |
| ManagerM1**("Fran", "Lamar", 4000.99, "Vice President");**  cout **<<** M1**.**toString**() <<** endl**;**  M1**.**setLastName**("P");**  cout **<<** M1**.**toString**() <<** endl**;**  M1**.**setLastName**("Prichard");**  cout **<<** M1**.**toString**() <<** endl**;** | -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Fran Lamar  Boss'es Title: Vice President  Salary: $4000.99  ----------------------------------------------  The new last name P is too short. There is no change.  -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Fran Lamar  Boss'es Title: Vice President  Salary: $4000.99  ----------------------------------------------  -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Fran Prichard  Boss'es Title: Vice President  Salary: $4000.99  ---------------------------------------------- |

If you have problems completing this step then see TA or instructor.

1. **Coding the mutator setFirstName for both classes.** In Employee class header file you just need one line of code for pure virtual function setFirstName. No code for this function is needed in Employee.cpp file. Then you need a corresponding pure virtual function **overriding code** in header and implementation files of Manager class. Notice that setFirstName function in Manager class should also enforce the invariant that first name cannot be less than three characters. If that is the case then previous name (constructed by constructor) is NOT changed. See details of setFirstName on pages 5 and 8. In the box below, write the algorithm for setFirstName function for Manager class.

Table below shows the source code and output to test the setFirstName function.

|  |  |
| --- | --- |
| Source code to test setFirstName function .[Do not copy and paste this source code into your program]. | Output from source code in left column |
| ManagerM1**("Fran", "Lamar", 4000.99, "Vice President");**  cout **<<** M1**.**toString**() <<** endl**;**  M1**.**setFirstName**("M");**  cout **<<** M1**.**toString**() <<** endl**;**  M1**.**setFirstName**("Lisa");**  cout **<<** M1**.**toString**() <<** endl**;** | -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Fran Lamar  Boss'es Title: Vice President  Salary: $4000.99  ----------------------------------------------  The new first name M is too short. There is no change.  -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Fran Lamar  Boss'es Title: Vice President  Salary: $4000.99  ----------------------------------------------  -------------------------------------------------  Manager's Data:  -------------------------------------------------  Name: Lisa Lamar  Boss'es Title: Vice President  Salary: $4000.99  ---------------------------------------------- |

If you have problems completing this step then see TA or instructor.

1. **Coding get functions or non-mutators for both classes:** Now complete the entry for pure virtual functions getFirstName, getLastName, getSalary, and getFullName in Employee class and override them in Manager class. Overridden versions in Manager class would have no more than 1 line of code. Their descriptions are given on pages: 4-5 and 7. The table below shows the test code for all three and output from that test code.

|  |  |
| --- | --- |
| Source code to test Pure Virtual get functions and their overridden versions. [Do not copy and paste source code below into Visual Studio]. | Output from source code in left column. |
| ManagerM1**("Fran", "Lamar", 4000.99, "Vice President");**  cout **<< "First Name: " <<** M1**.**getFirstName**() <<** endl**;**  cout **<< "Last Name: " <<** M1**.**getLastName**() <<** endl**;**  cout **<< "Full Name: " <<** M1**.**getFullName**() <<** endl**;** | First Name: Fran  Last Name: Lamar  Full Name: Fran Lamar |

1. **Coding remaining member functions of Manager class**

By now you have overridden all pure virtual functions of Employee class in its derived class Manager. Then next step is to complete the functions that are unique only to Manager class. These are: setSalary, setBossTitle, getSalary, and finally getBossTitle. We have no invariant related to salary except for the fact that it cannot be negative! Boss’s title cannot be less than 2 characters. This invariant is to be obeyed by the setBossTitle member function. If user attempts to set Boss’s title less than two characters, then no change in previous value is made. Please write by hand in the box below the algorithm for setBossTitle member function.

The table below shows the source code and output from it to test all four Manager class member functions.

|  |  |
| --- | --- |
| Source code to test rest of the four Manager class functions. [Please do not copy and paste this code in Visual Studio]. | Output from source code on left. |
| ManagerM1**("Fran", "Lamar", 4000.99, "Vice President");**  M1**.**setSalary**(-6000.55);**  M1**.**setBossTitle**("X");**  out **<< "Current Salary: $" <<** M1**.**getSalary**() <<** endl**;**  cout **<< "Current title of boss: " <<** M1**.**getBossTitle**() <<** endl**;**  M1**.**setBossTitle**("Executive Manager");**  cout **<< "Current title of boss: " <<** M1**.**getBossTitle**() <<** endl**;** | The boss's title cannot be less than 2 characters.  No change is made.  Current Salary: $6000.55  Current title of boss: Vice President  Current title of boss: Executive Manager |

**Main Function**

Main function presents following choices to the user to use the software built by Employee and Manager class:

"\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

"Enter the menu item of choice and press enter key."

"1. Create Manager instance from keyboard:"

"2. Print Manager to Console."

"3. Change Manager's last name."

"4. Change Manager's Salary."

"5. Change Manager's Boss'es Title."

"6. Print only Manager's name and Salary to console."

"7. Print only Manager's boss'es title to console."

"8. Exit";

**Additional requirement is that if user has not used menu choice 1 and directly proceeds to use the menu choices, 2 to 7, and then program asks the user to use menu choice 1 first. The output examples below show this requirement.**

|  |
| --- |
| Output Example showing enforcement of use of menu item 1 before using dependent menus 2 to 7  User input is in bold italic number. |
| *\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *Enter the menu item of choice and press enter key.*  *1. Create Manager instance from keyboard:*  *2. Print Manager to Console.*  *3. Change Manager's last name.*  *4. Change Manager's Salary.*  *5. Change Manager's Boss'es Title.*  *6. Print only Manager's name and Salary to console.*  *7. Print only Manager's boss'es title to console.*  *8. Exit*  ***2***  *Please create a Manager instance using option 1 before using this option.*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *Enter the menu item of choice and press enter key.*  *1. Create Manager instance from keyboard:*  *2. Print Manager to Console.*  *3. Change Manager's last name.*  *4. Change Manager's Salary.*  *5. Change Manager's Boss'es Title.*  *6. Print only Manager's name and Salary to console.*  *7. Print only Manager's boss'es title to console.*  *8. Exit*  ***3***  *Please create a Manager instance using option 1 before using this option.*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *Enter the menu item of choice and press enter key.*  *1. Create Manager instance from keyboard:*  *2. Print Manager to Console.*  *3. Change Manager's last name.*  *4. Change Manager's Salary.*  *5. Change Manager's Boss'es Title.*  *6. Print only Manager's name and Salary to console.*  *7. Print only Manager's boss'es title to console.*  *8. Exit*  ***4***  *Please create a Manager instance using option 1 before using this option.*  *\*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**  *Enter the menu item of choice and press enter key.*  *1. Create Manager instance from keyboard:*  *2. Print Manager to Console.*  *3. Change Manager's last name.*  *4. Change Manager's Salary.*  *5. Change Manager's Boss'es Title.*  *6. Print only Manager's name and Salary to console.*  *7. Print only Manager's boss'es title to console.*  *8. Exit* |

The Table below shows portions of normal main function output.

|  |
| --- |
| Output Example 2  User input is in bold italic number. |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***1***  Enter first name of Manager: John  Enter last name of Manager: Doe  Enter title of Manager's Boss: Director  Enter Manager's salary [xx.yy] 2000.99  \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***2***  Manager's Data:  Name: John Doe  Boss'es Title: Director  Salary: $2000.99  ----------------------------------------------  \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***3***  Enter Manager's new last name: Moe  \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***2***  Manager's Data:  Name: John Moe  Boss'es Title: Director  Salary: $2000.99  ----------------------------------------------  \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***4***  Type manager's new salary and press enter key: 3000.99  \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***2***  Manager's Data:  Name: John Moe  Boss'es Title: Director  Salary: $3000.99  ----------------------------------------------  \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit |

|  |
| --- |
| **Output Example 3: When not enough characters are entered for first name, last name and Boss’s title.** |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\* Main Menu\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Enter the menu item of choice and press enter key.  1. Create Manager instance from keyboard:  2. Print Manager to Console.  3. Change Manager's last name.  4. Change Manager's Salary.  5. Change Manager's Boss'es Title.  6. Print only Manager's name and Salary to console.  7. Print only Manager's boss'es title to console.  8. Exit  ***1***  Enter first name of Manager: a  Enter last name of Manager: b  Enter title of Manager's Boss: c  Enter Manager's salary [xx.yy] 2000.99  The first name a does not have enough characters.  We will set first name to a default value of "First name not set."  The last name b does not have enough characters.  We will set last name to a default value of "Last name not set."  Title of boss cannot be less than 3. The default value is used. |