Table of Contents

1.0 Task A – Enhance Product Management System

1.1 Source Code and Explanation in Comment

1.1.1 Main.java

```
package Assignment2;

import java.util.InputMismatchException; // For try catch

import java.util.Scanner; // To scan input

public class Main {

	public static ProductManagementSystem pms = new ProductManagementSystem(); // To use method in ProductManagementSystem.java

	public static Scanner sc = new Scanner(System.in);  // Declare scanner

	public static void main (String[]args) // Arguments in main method

	{

		int selection=0; // Selection of Manager's input in main menu

		boolean validSelection=false, exitProgram=false;

		Menu menus =  new Menu(); // To use method in Menu.java

		while(!exitProgram) //If manager does not chose to exit program, the program will keep looping

	{

			System.out.println("Welcome to the product management system.\n");

			menus.menu1(); //Show Product Code Table

	do {

	   try{ //catch mismatch input error and other possible errors

	   menus.menu2(); //Show main menu

	   selection = sc.nextInt();

	   // Validation for Selection

	   if (selection < 1 || selection > 6)

	   {

		System.out.println("Invalid selection, please input between 1 and 6.");

		validSelection=false;

		menus.menu2();//Show main menu

		selection = sc.nextInt();
```

```
        }
        else
        {
                validSelection=true;
        }
        }
        catch(InputMismatchException e)
        {
                System.out.println("Invalid selection, please input an integer number.");
                validSelection = false;
                        sc.next();
        }
        catch(Exception e)
                {
                        System.out.println("Something is error.");
                }
                }while(!validSelection); //The menu will keep looping if the manager does not
input valid selection
      switch (selection)
      {
      case 1: //Manager chose to add new product
         menus.menu1(); //Show Product Code Table
         pms.addProduct();
         break;
      case 2: // Manager chose to update product
         pms.updateProduct();
         break;
      case 3: // Manager chose to delete product
         pms.deleteProduct();
         break;
```

```
        case 4: //Manager chose to display product based on different criteria
            pms.displayProduct();
            break;
        case 5: //Manager chose to display product code table
            menus.menu1();
            break;
        case 6: //Manager chose to exit the program
            System.out.println("Shutting down the system. Have a nice day.");
            exitProgram=true;
            break;
        default:
            System.out.println("Please enter valid selection.");
        }
            }
            }
}
```

1.1.2 ProductManagementSystem.java

```
package Assignment2;

import java.util.ArrayList; // Data structure chose

import java.util.Collections;

import java.util.Scanner; // To scan input

import java.util.InputMismatchException; //For try catch

import java.util.Comparator; //For sorting

public class ProductManagementSystem {

        private static Scanner sc = new Scanner(System.in); //Declare scanner

        private ArrayList<Product> products = new ArrayList<>(); //Declare arraylist

        private Menu menus =  new Menu(); //To use method in Menu.java


        public ProductManagementSystem() //For initially store product records

        {

      products = new ArrayList<>(); //Declare product arraylist

      // Add sample data as it is more realistic and more convenient in testing features


      products.add(new Product("J53201024","Japan", "Intel i5","320 GB", "1 TB", 5));

      products.add(new Product("A53201024","America", "Intel i5", "320 GB", "1 TB", 20));

      products.add(new Product("M73201024","Malaysia", "Intel i7", "320 GB", "1 TB", 10));

      products.add(new Product("J95002048","Japan", "Intel i9","500 GB", "2 GB", 5));

      products.add(new Product("A510244096","America", "Intel i5", "1 TB", "4 GB", 20));

      products.add(new Product("M55002048","Malaysia", "Intel i5", "500 GB", "2 GB", 10));

      products.add(new Product("J510242048","Japan", "Intel i5","1 TB", "2 GB", 5));

      products.add(new Product("M910244096","Malaysia", "Intel i9", "1 TB", "4 GB", 20));

      products.add(new Product("A910241024","America", "Intel i9", "1 TB", "1 TB", 10));

  }


        public void addProduct() //(1) Add New Products

        {
```

```
		boolean productCodeDuplication = false; //validate if the product code already exist in the system
		boolean validateCountry = false; //validate if the country of the product code is correct such as 'M'
		boolean validateProcessor = false; //validate if the processor type is correct such as '5'
		boolean validateHardDisk = false; //validate if the hard disk is correct such as '320'
		boolean validateInternalMemory = false; //validate if the internal memory is correct such as '1024'
		boolean validateQuantity = false; //validate the quantity is correct as it is positive and numeric
		boolean validateSelection = false; //validate if the selection is in the option list and numeric
		String productCode = "";
		int selection=0;
		do{
	try{ //catch possible error
	try { //catch input mismatch error for selection
	System.out.println("Enter your selection.");
	menus.menu12(); // to chose either add record or exit
	selection = sc.nextInt();
	while(selection!=1 && selection!=2) //validate selection in range
	{
	   System.out.println("Please enter either 1 or 2.");
	   System.out.println("Enter your selection.");
	   menus.menu12(); // ask to choose add record or exit
	   selection = sc.nextInt();
	}
	   }catch(InputMismatchException e)
	   {
		System.out.println("Invalid selection, please input an integer number.");
```

```
                    validateSelection=false;
                    sc.next();
            }
        switch (selection)
        {
        case 1: //(1) Add Records
        do
        {
            Product product = new Product();
        validateSelection = true;
        sc.nextLine(); //Consume new line
        System.out.println("Enter the product code of the new product according to the product code table. ");
        System.out.print("Product code: ");
        productCode = sc.nextLine().toUpperCase(); //to handle both upper case and lower case country character (eg:M and m)
        while(productCode.isEmpty()) // validate it is not empty
        {
            System.out.println("The product code should not be empty.");
            System.out.println("Please try again.");
            System.out.println("Enter the product code of the new product according to the product code table. ");
            System.out.print("Product code: ");
            productCode = sc.nextLine().toUpperCase();
        }
        while(productCode.length() < 9 || productCode.length() > 10 ) //Validate the length of the product code
        {
            System.out.println("Invalid length. The product code should only consists of 9 - 10 characters.");
            System.out.println("Please try again.");
```

```
        System.out.println("Enter the product code of the new product according to the product code table. ");
        System.out.print("Product code: ");
        productCode = sc.nextLine().toUpperCase();
    }
    do
    {
    for (Product product_ : products)
    {
      if(product_.getProductCode().equals(productCode)) //validate if the code already exist
      {
        System.out.println("The records of the product code is already in the "
                    + "system, please enter unexisting product code.");
        productCodeDuplication = true;
        System.out.println("Please try again.");
        System.out.println("Enter the product code of the new product according to the product code table. ");
        System.out.print("Product code: ");
        productCode = sc.nextLine().toUpperCase();
      }
      else
      {
            productCodeDuplication=false;
      }
    }
    }while(productCodeDuplication);
    String stringProductCode = productCode;
    //Retrieve the records details from the product code using charAt
    char countryCharacter = stringProductCode.charAt(0); //country character of the product code at index 0
```

```
        char processorCharacter = stringProductCode.charAt(1); //processor character of the
product code at index 0
        char hardDiskCharacter = stringProductCode.charAt(2); //hard disk character of the
product code at index 0
        char internalMemoryCharacter=stringProductCode.charAt(5); //default product code length
9
        if (productCode.length()==10) /*if the length of the product code is 10, the
           internal memory character will start at index of 6*/
        {
           internalMemoryCharacter = stringProductCode.charAt(6);
        }

        //Set country according to product code
        switch (countryCharacter)
        {
           case 'M':
              product.setCountry("Malaysia");
              validateCountry = true;
              break;

           case 'J':
              product.setCountry("Japan");
              validateCountry = true;
              break;

           case 'A':
              product.setCountry("America");
              validateCountry = true;
              break;
           default:
              System.out.println("Invalid country.");
```

```
            validateCountry = false;
        }
        //Set processor type according to product code
        switch(processorCharacter) {
            case '5':
                product.setProcessor("Intel i5");
                validateProcessor = true;
                break;

            case '7':
                product.setProcessor("Intel i7");
                validateProcessor = true;
                break;

            case '9':
                product.setProcessor("Intel i9");
                validateProcessor = true;
                break;

            default:
                System.out.println("Invalid processor type.");
                validateProcessor = false;
        }
        //hard disk character start from index 2 to index 4
        String hardDiskChar = productCode.substring(2,5);
      //Set hard disk capacity according to product code
        switch(hardDiskCharacter) {
            case '3':

                //Validate is 320 correctly typed
```

```
            if(hardDiskChar.equals("320"))
            {
               product.setHardDiskCapacity("320 GB");
               validateHardDisk = true;
               break;
            }
            else
            {
               System.out.println("The hard disk capacity should be 320. Please enter again.");
               validateHardDisk = false;
               break;
            }



        case '5':
                  //Validate is 500 correctly typed
            if(hardDiskChar.equals("500"))
            {
               product.setHardDiskCapacity("500 GB");
               validateHardDisk = true;
               break;
            }
            else
            {
               System.out.println("The hard disk capacity should be 500. Please enter again. ");
               validateHardDisk = false;
               break;
            }
```

```
            case '1':
                String hardDiskChar2 = productCode.substring(2,6);
                //Validate is 1024 correctly typed
               if(hardDiskChar2.equals("1024"))
               {
                  product.setHardDiskCapacity("1 TB");
                  validateHardDisk = true;
                  break;
               }
               else
               {
                  System.out.println("The hard disk capacity should be 1024. Please enter again.
");
                  validateHardDisk = false;
                  break;
               }



         default:
            //If not equal to either 320, 500, 1024
            System.out.println("Invalid hard disk capacity.");
            validateHardDisk = false;
      }
      //Set internal memory capacity according to product code
      switch(internalMemoryCharacter) {
         case '1':
                        // If the length of the product code is 9, then the internal memory
character start from index 6
                  if(productCode.length() == 9)
                  {
```

```
                    String memoryCharacter = productCode.substring(5);
                    //Validate is 1024 correctly typed
                    if(memoryCharacter.equals("1024")){
                        product.setInternalMemoryCapacity("1 TB");
                        validateInternalMemory = true;
                        break;
                    }
                    else
                    {
                        System.out.println("The internal memory capacity should be 1024. Please enter again.");
                        validateInternalMemory = false;
                        break;
                    }
            }

             // If the length of the product code is 10, then the internal memory character start from index 6
            else if(productCode.length() == 10){
                String memoryCharacter = productCode.substring(6);
                //Validate is 1024 correctly typed
                if(memoryCharacter.equals("1024")){
                    product.setInternalMemoryCapacity("1 TB");
                    validateInternalMemory = true;
                    break;
                }else{
                    System.out.println("The internal memory capacity should be 1024. Please enter again.");
                    validateInternalMemory = false;
                    break;
                }
```

```
                }
            case '2':
                // If the length of the product code is 9, then the internal memory character start from index 6
                if(productCode.length() == 9)
                {
                        String memoryCharacter = productCode.substring(5);
                        //Validate is 2048 correctly typed
                        if(memoryCharacter.equals("2048")){
                                product.setInternalMemoryCapacity("2 GB");
                                validateInternalMemory = true;
                                break;
                        }
                        else
                        {
                                System.out.println("The internal memory capacity should be 2048. Please enter again.");
                                validateInternalMemory = false;
                                break;
                        }
            }

             // If the length of the product code is 10, then the internal memory character start from index 6
            else if(productCode.length() == 10){
                String memoryCharacter = productCode.substring(6);
                //Validate is 2048 correctly typed
                if(memoryCharacter.equals("2048")){
                                product.setInternalMemoryCapacity("2 GB");
                    validateInternalMemory = true;
                    break;
```

```
                }else{
                    System.out.println("The internal memory capacity should be 2048. Please enter
again.");
                    validateInternalMemory = false;
                    break;
                }
            }
            break;
        case '4':
                // If the length of the product code is 9, then the internal memory character start
from index 6
                if(productCode.length() == 9)
                {
                        String memoryCharacter = productCode.substring(5);
                        //Validate is 4096 correctly typed
                        if(memoryCharacter.equals("4096")){
                                product.setInternalMemoryCapacity("4 GB");
                                validateInternalMemory = true;
                                break;
                        }
                        else
                        {
                                System.out.println("The internal memory capacity should be 4096.
Please enter again.");
                                validateInternalMemory = false;
                                break;
                        }
            }


             // If the length of the product code is 10, then the internal memory character start from
index 6
```

```
            else if(productCode.length() == 10){
                String memoryCharacter = productCode.substring(6);
                //Validate is 4096 correctly typed
                if(memoryCharacter.equals("4096")){
                                product.setInternalMemoryCapacity("4 GB");
                    validateInternalMemory = true;
                    break;
                }else
                {
                    System.out.println("The internal memory capacity should be 4096. Please enter
again.");
                    validateInternalMemory = false;
                    break;
                }
            }
            break;
        default :
            System.out.println("Invalid internal memory capacity.");
            validateInternalMemory = false;
    }
    //only run if the product code is totally validate
    if(validateSelection && !productCodeDuplication && validateCountry && validateProcessor
&&
                validateHardDisk && validateInternalMemory && validateSelection)
                {
    do
    {
                try //catch input mismatch errors and other possible errors
                {
                //prompt user to input the quantity for the new record
                System.out.print("Enter product quantity: ");
```

```
                int quantity = sc.nextInt();
                while(quantity<0) //validate quantity should be positive
                {
                        System.out.println("Invalid quantity number. It should be positive, please
enter again.");
                        System.out.print("Enter product quantity: ");
                        quantity = sc.nextInt();
                }
                validateQuantity = true;
                product.setProductCode(productCode); //set product code according to user
input
                product.setQuantity(quantity); //set quantity according to user input
        products.add(product); //add product
        System.out.println("Product record successfully added.");
                }
                catch(InputMismatchException e)
                {
                        System.out.println("Invalid selection, please input an integer number.");
                        validateQuantity=false;
                        sc.next();
                }
                catch(Exception e)
                {
                        System.out.println("Something is error.");
                }
        }while(!validateQuantity);
                }
        //prompt user for not adding the product as it is invalid
        if(productCodeDuplication || !validateCountry || !validateProcessor ||
                !validateHardDisk || !validateInternalMemory || !validateQuantity
|| !validateSelection)
```

```
        {
            System.out.println("Product failed to add, please check again the product code.");
        }
        break;
        }while(productCodeDuplication || !validateCountry || !validateProcessor ||
                    !validateHardDisk || !validateInternalMemory || !validateQuantity
|| !validateSelection);


        case 2: //(2) Exit
            validateSelection = true;
            break;
        }
        }
        catch(Exception e)
                    {
                        System.out.println("Something is error.");
                    }
    }while(!validateSelection || selection!=2);
                    //will keep looping if either selection is invalid or user not chose to exit
            }


            public void updateProduct() //(2) Update Records
            {
                    boolean validSelection = false;
                    int selection=0;
                    do
                    {
                    try // catch input mismatch error and other possible errors
                    {
                    boolean validSelection2=false;
```

```
            menus.menu8();
            selection = sc.nextInt();
            switch(selection)
            {
            case 1: // Update (1) Manufacturing Country
                  do
                  {
                  try //catch mismatch input error and other possible errors
                  {
                  int selecCount1=0, selecCount2=0, updateSelection = 0; /*selecCount1 is old
country option
                  selecCount2 is new country option*/
                  int count=0; //to calculate how many records equal to the old country
                  String originalCountry="", newCountry="";
                  do
                  {
                  do
                  {
                  System.out.println("Select the manufacturing country of the product you want to
update.");
                  menus.menu4(); //display country for user to select
                  selecCount1 = sc.nextInt();
                  if(selecCount1<1||selecCount1>3)
                  {
                        System.out.println("Invalid selection. Please enter between 1 - 3.");
                  }
                  }while(selecCount1<1||selecCount1>3);
                  do
                  {
                  System.out.println("Select the new manufacturing country:");
                  menus.menu4();
```

```
selecCount2 = sc.nextInt();
if(selecCount2<1||selecCount2>3)
{
	System.out.println("Invalid selection. Please enter between 1 - 3.");
}
}while(selecCount2<1||selecCount2>3);
if (selecCount1==selecCount2) //validate if old and new is same
{
	System.out.println("The old and new manufacturing country should not be the same. Please try again.");
}
}while(selecCount1==selecCount2);
switch(selecCount1) //assign original country according to option
{
case 1:
	originalCountry = "Malaysia";
	break;

case 2:
	originalCountry = "Japan";
	 break;

case 3:
	originalCountry = "America";
	break;

default:
	System.out.println("Invalid selection. Please try again.");
	validSelection2 = false;
}
```

```
                switch(selecCount2) //assign new country according to option
                {
                case 1:
                        newCountry = "Malaysia";
                        break;

                case 2:
                        newCountry = "Japan";
                         break;

                case 3:
                        newCountry = "America";
                        break;

                default:
                        System.out.println("Invalid selection. Please try again.");
                        validSelection2 = false;
                }
                for (Product product : products) //count number of records same as original
country
        {
                if(product.getCountry().equals(originalCountry))
           {
              count++;
           }
        }
                System.out.println("There are "+count+" records with manufacturing country of
"+originalCountry+":");
                if (count!=0) // if at least one record will display the record that is old country
                {
                menus.menu(); //print header
```

```
                for (Product product : products) //display records that are same as original
country
        {
                if(product.getCountry().equals(originalCountry))
           {
              System.out.print(product);
           }
        }
                menus.menu9(); //ask either update all or update one
                updateSelection = sc.nextInt();
                switch(updateSelection)
                {
                case 1: // (1) Update All
                        for (Product product : products) //set every product that are original
country to new country
                    {
                       if (product.getCountry().equals(originalCountry))
                       {
                          product.setCountry(newCountry);
                       }
                    }
                        System.out.println("The product records has been update.");
                        break;

                case 2: // (2) Update One
                        int cont=0; //for user to continue update or not
                        do
                        {
                        Product productX = null; // to store the product
                        String productCode="";
                        boolean validProductCode = true, validProduct = true;
```

```
                    do
                    {
                        System.out.println("Please enter the product code you want to
update.");
                        System.out.print("Product code: ");
                        if(validProductCode && validProduct)
                        {
                            sc.nextLine(); //Consume new line
                        }
                        productCode = sc.nextLine().toUpperCase();
                        for (Product product : products) //validate if the product is in the
system
                    {
                       if (!product.getProductCode().equals(productCode))
                       {
                          validProductCode = false;
                       }
                       else
                       {
                          productX = product;
                          validProductCode = true;
                              break;
                       }
                    }
                        //if the product code is validate but the product country is already
the new country
                        if (validProductCode && productX != null &&
productX.getCountry().equals(newCountry))
                        {
                            System.out.println("The country of "+productCode+" is
already "+newCountry+".");
                            validProduct = false;
```

```
                                }
                                else
                                {
                                        validProduct = true;
                                }
                                if(!validProductCode)
                                {
                                        System.out.println("Product code not exist, please add the
product to the system first.");
                                }
                        }while(!validProductCode || !validProduct); //will keep looping if the
product or product code invalid
                        for (Product product : products) //update the product country to new
country
                {
                                if (product.getProductCode().equals(productCode))
                   {
                      product.setCountry(newCountry);
                   }
                }
                        System.out.println(productCode+" successfully updated.");
                        System.out.println("Continue update?");
                        menus.menu10(); //ask user to select yes or no
                        cont = sc.nextInt();
                        }while(cont!=2); //will keep looping until user choose no for the continue
option
                        break;

                default:
                        System.out.println("Invalid selection. Please try again.");
                        validSelection2 = false;
```

```
                }
                }
                validSelection2=true;
                }catch(InputMismatchException e)
                        {
                                System.out.println("Invalid selection, please input an integer
number.");
                                validSelection=false;
                                sc.next();
                        }
                catch(Exception e)
                        {
                                System.out.println("Something is error.");
                        }
                        }while(!validSelection2);
                break;


        case 2: // Update (2) Processor Type
                do
                {
                try //catch mismatch input error and other possible errors
                {
                int selecCount1=0, selecCount2=0, updateSelection = 0; //selecCount1 is old
option selecCount2 is new option
                int count=0; //count product with original processor
                String originalProcessor="", newProcessor="";
                do
                {
                do
                {
```

```
            System.out.println("Select the processor type of the product you want to
update.");
            menus.menu5(); //ask user to choose original processor type i5 i7 i9
            selecCount1 = sc.nextInt();
            if(selecCount1<1||selecCount1>3)
            {
                  System.out.println("Invalid selection. Please enter between 1 - 3.");
            }
            }while(selecCount1<1||selecCount1>3);
            do
            {
            System.out.println("Select the new processor type:");
            menus.menu5(); //ask user to choose new processor type i5 i7 i9
            selecCount2 = sc.nextInt();
            if(selecCount2<1||selecCount2>3)
            {
                  System.out.println("Invalid selection. Please enter between 1 - 3.");
            }
            }while(selecCount2<1||selecCount2>3);
            if (selecCount1==selecCount2) //validate if original and new is same option
            {
                  System.out.println("The old and new processor type should not be the
same. Please try again.");
            }
            }while(selecCount1==selecCount2);
            switch(selecCount1) //assign original according to option
            {
            case 1:
                  originalProcessor = "Intel i5";
                  break;
```

```
case 2:
        originalProcessor = "Intel i7";
         break;

case 3:
        originalProcessor = "Intel i9";
        break;

default:
        System.out.println("Invalid selection. Please try again.");
        validSelection2 = false;
}
switch(selecCount2) //assign new according to option
{
case 1:
        newProcessor = "Intel i5";
        break;

case 2:
        newProcessor = "Intel i7";
         break;

case 3:
        newProcessor = "Intel i9";
        break;

default:
        System.out.println("Invalid selection. Please try again.");
        validSelection2 = false;
}
```

```
                for (Product product : products) //count product same as original processor
        {
                if(product.getProcessor().equals(originalProcessor))
           {
              count++;
           }
        }
                System.out.println("There are "+count+" records with processor type of
"+originalProcessor+":");
                if (count!=0)
                {
                menus.menu(); //print header
                for (Product product : products) //display record with same original processor
        {
                        if(product.getProcessor().equals(originalProcessor))
           {
              System.out.print(product);
           }
        }
                menus.menu9(); //ask update all or one
                updateSelection = sc.nextInt();
                switch(updateSelection)
                {
                case 1: // (1) Update All
                        for (Product product : products)
                   {
                                if(product.getProcessor().equals(originalProcessor))
                      {
                         product.setProcessor(newProcessor);
                      }
```

```
                }
                        System.out.println("The product records has been update.");
                        break;
                case 2:
                        int cont=0; // for user choose to continue or not
                        do
                        {
                                Product productX = null; //store product
                                String productCode="";
                                boolean validProductCode = true, validProduct = true;
                        do
                        {
                                System.out.println("Please enter the product code you want to
update.");
                                System.out.print("Product code: ");
                                if(validProductCode && validProduct)
                                {
                                        sc.nextLine(); //Consume new line
                                }
                                productCode = sc.nextLine().toUpperCase();
                                for (Product product : products) //check if product code is validate
                           {
                              if (!product.getProductCode().equals(productCode))
                              {
                                 validProductCode = false;
                              }
                              else
                              {
                                 productX = product;
                                 validProductCode = true;
```

```
                                    break;
                                }
                            }
                                //if product code is validate but old same as new
                                if (validProductCode && productX != null &&
productX.getProcessor().equals(newProcessor))
                                {
                                    System.out.println("The processor type of
"+productCode+" is already "+newProcessor+".");
                                    validProduct = false;
                                }
                                else
                                {
                                    validProduct = true;
                                }
                                if(!validProductCode)
                                {
                                    System.out.println("Product code not exist, please add the
product to the system first.");
                                }
                            }while(!validProductCode || !validProduct);
                            for (Product product : products) //set product to new processor
                        {
                                if (product.getProductCode().equals(productCode))
                           {
                              product.setProcessor(newProcessor);
                           }
                        }
                            System.out.println(productCode+" successfully updated.");
                            System.out.println("Continue update?");
                            menus.menu10();
```

```
                cont = sc.nextInt();
                }while(cont!=2); //will keep looping until user choose no for continue
option
                break;

            default:
                System.out.println("Invalid selection. Please try again.");
                validSelection2 = false;
            }
            }
            validSelection2 = true;
            }catch(InputMismatchException e)
                {
                    System.out.println("Invalid selection, please input an integer
number.");
                    validSelection=false;
                    sc.next();
                }
            catch(Exception e)
                {
                    System.out.println("Something is error.");
                }
                }while(!validSelection2);
            break;

        case 3: // Update (3) Hard Disk Capacity
            do
            {
            try //catch mismatch input error and other possible errors
            {
```

```
		int selecCount1=0, selecCount2=0, updateSelection = 0; //selecCount1 is old hard disk selecCount2 is new
		int count=0; //count product with same original hard disk
		String originalHardDisk="", newHardDisk="";
		do
		{
		do
		{
		System.out.println("Select the hard disk capacity of the product you want to update.");
		menus.menu6(); //ask to select old capacity 320 500 1024
		selecCount1 = sc.nextInt();
		if(selecCount1<1||selecCount1>3)
		{
			System.out.println("Invalid selection. Please enter between 1 - 3.");
		}
		}while(selecCount1<1||selecCount1>3);
		do
		{
		System.out.println("Select the new hard disk capacity:");
		menus.menu6(); //ask to select new capacity 320 500 1024
		selecCount2 = sc.nextInt();
		if(selecCount2<1||selecCount2>3)
		{
			System.out.println("Invalid selection. Please enter between 1 - 3.");
		}
		}while(selecCount2<1||selecCount2>3);
		if (selecCount1==selecCount2) //check if old and new is same
		{
			System.out.println("The old and new hard disk capacity should not be the same. Please try again.");
```

```
            }
            }while(selecCount1==selecCount2);
            switch(selecCount1) //assign old hard disk according to option
            {
            case 1:
                originalHardDisk = "320 GB";
                break;

            case 2:
                originalHardDisk = "500 GB";
                 break;

            case 3:
                originalHardDisk = "1 TB";
                break;

            default:
                System.out.println("Invalid selection. Please try again.");
                validSelection2 = false;
            }
            switch(selecCount2) //assign new hard disk according to option
            {
            case 1:
                newHardDisk = "320 GB";
                break;

            case 2:
                newHardDisk = "500 GB";
                 break;
```

```
                case 3:
                        newHardDisk = "1 TB";
                        break;


                default:
                        System.out.println("Invalid selection. Please try again.");
                        validSelection2 = false;


                }
                for (Product product : products) //count product same as original hard disk
        {
                if(product.getHardDiskCapacity().equals(originalHardDisk))
           {
              count++;
           }
        }
                System.out.println("There are "+count+" records with hard disk capacity of
"+originalHardDisk+":");
                if (count!=0)
                {
                menus.menu(); //print header
                for (Product product : products) //display product with same hard disk as original
        {
                        if(product.getHardDiskCapacity().equals(originalHardDisk))
           {
              System.out.print(product);
           }
        }
                menus.menu9(); //ask to update all or one
                updateSelection = sc.nextInt();
```

```
                switch(updateSelection)
                {
                case 1: //(1) Update All
                        for (Product product : products)
                  {
                                if(product.getHardDiskCapacity().equals(originalHardDisk))
                     {
                        product.setHardDiskCapacity(newHardDisk);
                     }
                  }
                        System.out.println("The product records has been update.");
                        break;
                case 2:
                        int cont=0; //for user to choose whether continue update or not
                        do
                        {
                                Product productX = null; //to store product
                                String productCode="";
                                boolean validProductCode = true, validProduct = true;
                        do
                        {
                                System.out.println("Please enter the product code you want to
update.");
                                System.out.print("Product code: ");
                                if(validProductCode && validProduct)
                                {
                                        sc.nextLine(); //Consume new line
                                }
                                productCode = sc.nextLine().toUpperCase();
                                for (Product product : products)  //validate productCode exist or
not
```

```
                {
                    if (!product.getProductCode().equals(productCode))
                    {
                        validProductCode = false;
                    }
                    else
                    {
                        productX = product;
                        validProductCode = true;
                            break;
                    }
                }
                        // if productCode valid but original same with new hard disk
                        if (validProductCode && productX != null &&
productX.getHardDiskCapacity().equals(newHardDisk))
                        {
                                System.out.println("The hard disk capacity of
"+productCode+" is already "+newHardDisk+".");
                                validProduct = false;
                        }
                        else
                        {
                                validProduct = true;
                        }
                        if(!validProductCode)
                        {
                                System.out.println("Product code not exist, please add the
product to the system first.");
                        }
            }while(!validProductCode || !validProduct);
            for (Product product : products)  //set product hard disk to new hard disk
```

```
                {
                        if (product.getProductCode().equals(productCode))
                   {
                      product.setHardDiskCapacity(newHardDisk);
                   }
                }
                      System.out.println(productCode+" successfully updated.");
                      System.out.println("Continue update?");
                      menus.menu10(); //ask user yes or no
                      cont = sc.nextInt();
                      }while(cont!=2); //will keep looping until user choose exit option
                      break;

            default:
                      System.out.println("Invalid selection. Please try again.");
                      validSelection2 = false;
            }
            }
            validSelection2 = true;
            }catch(InputMismatchException e)
                      {
                              System.out.println("Invalid selection, please input an integer
number.");
                              validSelection=false;
                              sc.next();
                      }
            catch(Exception e)
                      {
                              System.out.println("Something is error.");
                      }
```

```
                }while(!validSelection2);
            break;


        case 4: // Update (4) Internal Memory Capacity
            do
            {
            try
            {
            int selecCount1=0, selecCount2=0, updateSelection = 0; //selecCount1 is old capacity option selecCount2 is new
            int count=0; //for counting numbers of product same as original capacity
            String originalInternalMemory="", newInternalMemory="";
            do
            {
            do
            {
            System.out.println("Select the internal memory capacity of the product you want to update.");
            menus.menu7(); //ask user to select old capacity
            selecCount1 = sc.nextInt();
            if(selecCount1<1||selecCount1>3)
            {
                System.out.println("Invalid selection. Please enter between 1 - 3.");
            }
            }while(selecCount1<1||selecCount1>3);
            do
            {
            System.out.println("Select the new internal memory capacity:");
            menus.menu7(); //ask user to select new capacity
            selecCount2 = sc.nextInt();
            if(selecCount2<1||selecCount2>3)
```

```
{
        System.out.println("Invalid selection. Please enter between 1 - 3.");
}
}while(selecCount2<1||selecCount2>3);
if (selecCount1==selecCount2) //validate if old and new option are the same
{
        System.out.println("The old and new internal memory capacity should not be the same. Please try again.");
}
}while(selecCount1==selecCount2);
switch(selecCount1) //assign old capacity according to option
{
case 1:
        originalInternalMemory = "1 TB";
        break;

case 2:
        originalInternalMemory = "2 GB";
         break;

case 3:
        originalInternalMemory = "4 GB";
        break;

default:
        System.out.println("Invalid selection. Please try again.");
        validSelection2 = false;
}
switch(selecCount2) //assign new capacity according to option
{
```

```
                case 1:
                        newInternalMemory = "1 TB";
                        break;

                case 2:
                        newInternalMemory = "2 GB";
                         break;

                case 3:
                        newInternalMemory = "4 GB";
                        break;

                default:
                        System.out.println("Invalid selection. Please try again.");
                        validSelection2 = false;
                }
                for (Product product : products) //count product same as original internal memory
capacity
        {
                if(product.getInternalMemoryCapacity().equals(originalInternalMemory))
           {
              count++;
           }
        }
                System.out.println("There are "+count+" records with internal memory capacity of
"+originalInternalMemory+":");
                if (count!=0)
                {
                menus.menu(); //print headers
                for (Product product : products) //display product same as original internal
memory
```

```
{
                if(product.getInternalMemoryCapacity().equals(originalInternalMemory))
   {
      System.out.print(product);
   }
}
          menus.menu9(); //ask to update all or one
          updateSelection = sc.nextInt();
          switch(updateSelection)
          {
          case 1: //(1) Update All
                for (Product product : products)
            {

if(product.getInternalMemoryCapacity().equals(originalInternalMemory))
                {
                   product.setInternalMemoryCapacity(newInternalMemory);
                }
            }
                System.out.println("The product records has been update.");
                break;


          case 2:
                int cont=0; //for user to choose whether continue or not
                do
                {
                        Product productX = null; //to store product
                        String productCode="";
                        boolean validProductCode = true, validProduct = true;
                do
```

```
                {
                        System.out.println("Please enter the product code you want to
update.");
                        System.out.print("Product code: ");
                        if(validProductCode && validProduct)
                        {
                                sc.nextLine(); //Consume new line
                        }
                        productCode = sc.nextLine().toUpperCase();
                        for (Product product : products)  //to validate if product code exist
                    {
                       if (!product.getProductCode().equals(productCode))
                       {
                          validProductCode = false;
                       }
                       else
                       {
                          productX = product;
                          validProductCode = true;
                                break;
                       }
                    }
                        //if product code exist but original memory same as new memory
                        if (validProductCode && productX != null &&
productX.getInternalMemoryCapacity().
                                        equals(newInternalMemory))
                        {
                                System.out.println("The internal memory capacity of
"+productCode+" is already "+newInternalMemory+".");
                                validProduct = false;
                        }
```

```
                                else
                                {
                                        validProduct = true;
                                }
                                if(!validProductCode)
                                {
                                        System.out.println("Product code not exist, please add the
product to the system first.");
                                }
                        }while(!validProductCode || !validProduct);
                        for (Product product : products)  //set product memory to new memory
                    {
                                if (product.getProductCode().equals(productCode))
                       {
                         product.setInternalMemoryCapacity(newInternalMemory);
                       }
                    }
                        System.out.println(productCode+" successfully updated.");
                        System.out.println("Continue update?");
                        menus.menu10(); //ask user to choose yes or no
                        cont = sc.nextInt();
                        }while(cont!=2); //will keep looping until user choose no for continue
option
                        break;
                default:
                        System.out.println("Invalid selection. Please try again.");
                        validSelection2 = false;
                }
                }
                validSelection2=true;
                }catch(InputMismatchException e)
```

```
                {
                        System.out.println("Invalid selection, please input an integer
number.");
                        validSelection=false;
                        sc.next();
                }
            catch(Exception e)
                {
                        System.out.println("Something is error.");
                }
                }while(!validSelection2);
            break;


        case 5: // Update (5) Quantity
            do
            {
            try
            {
                int quant1, quant2, updateSelection, count=0;
                /*quant1=old quantity
                 quant2=new
                update selection = update all or one
                count use to count product with same old quantity*/
                boolean validQuantity = false;
                do
                {
                do
                {
                System.out.println("Enter the quantity of the product you want to
update.");
                System.out.print("Quantity: ");
```

```
                    quant1 = sc.nextInt();
                    if(quant1<0)
                    {
                            System.out.println("Invalid quantity, the quantity should not be
negative.");
                    }
                    }while(quant1<0);
                    for (Product product : products) //check whether there are record with old
quantity
            {
                            if(product.getQuantity()!=quant1)
               {
                  validQuantity = false;
               }
                            else
                            {
                                    validQuantity = true;
                                    break;
                            }
            }
                    if (!validQuantity)
                    {
                            System.out.println("No records with quantity of "+quant1+". Please
enter again.");
                    }
                    }while(!validQuantity);
                    do
                    {
                    System.out.println("Enter the new quantity:");
                    System.out.print("Quantity: ");
                    quant2 = sc.nextInt();
```

```
                    if(quant2<0)
                    {
                        System.out.println("Invalid quantity, the quantity should not be
negative.");
                    }
                    }while(quant2<0);
                    if (quant1==quant2) //check if old and new quantity are the same
                    {
                        System.out.println("The old and new quantity should not be the
same. Please try again.");
                    }
                    for (Product product : products) //count for the product with old quantity
                {
                    if(product.getQuantity()==quant1)
                  {
                     count++;
                  }
                }
                    System.out.println("There are "+count+" records with quantity of
"+quant1+":");
                    if (count!=0)
                    {
                    menus.menu(); //print header
                    for (Product product : products) //display product with old quantity
                {
                        if(product.getQuantity()==quant1)
                  {
                     System.out.print(product);
                  }
                }
                    }
```

```
menus.menu9(); //ask to update all or one
updateSelection = sc.nextInt();
switch(updateSelection)
{
case 1: //(1) Update All
        for (Product product : products)
   {
            if(product.getQuantity()==quant1)
      {
         product.setQuantity(quant2);
      }
   }
        System.out.println("The product records has been update.");
        break;

case 2: //(2) Update one
        int cont=0; //use for user to choose continue update or not
        do
        {
            Product productX = null; //to store product
            String productCode="";
            boolean validProductCode = true, validProduct = true;
        do
        {
            System.out.println("Please enter the product code you
want to update.");
            System.out.print("Product code: ");
            if(validProductCode && validProduct)
            {
                sc.nextLine(); //Consume new line
```

```
                    }
                    productCode = sc.nextLine().toUpperCase();
                    for (Product product : products) //validate product code
exist or not
                {
                    if (!product.getProductCode().equals(productCode))
                    {
                        validProductCode = false;
                    }
                    else
                    {
                        productX = product;
                        validProductCode = true;
                            break;
                    }
                }
                    //if product code validate but its quantity same as new
quantity
                    if (validProductCode && productX != null &&
productX.getQuantity()==(quant2))
                    {
                            System.out.println("The quantity of
"+productCode+" is already "+quant2+".");
                            validProduct = false;
                    }
                    else
                    {
                            validProduct = true;
                    }
                    if(!validProductCode)
                    {
```

```
                                        System.out.println("Product code not exist, please
add the product to the system first.");
                                    }
                            }while(!validProductCode || !validProduct);
                            for (Product product : products) //set product quantity to new
quantity
                        {
                                if (product.getProductCode().equals(productCode))
                           {
                            product.setQuantity(quant2);
                           }
                        }
                            System.out.println(productCode+" successfully updated.");
                            System.out.println("Continue update?");
                            menus.menu10(); //ask for user to choose yes or no
                            cont = sc.nextInt();
                            }while(cont!=2); //will keep looping until user choose no for
continue option
                            break;
                    default:

                            System.out.println("Invalid selection. Please try again.");
                            validSelection2 = false;
                    }
            validSelection2=true;
            }catch(InputMismatchException e)
                    {
                            System.out.println("Invalid selection, please input an integer
number.");
                            validSelection=false;
                            sc.next();
```

```
                }
            catch(Exception e)
                {
                    System.out.println("Something is error.");
                }
                }while(!validSelection2);
            break;

        case 6: // (6) Exit Update
            validSelection = true;
            break;

        default:
    System.out.println("Please enter valid selection.");
        }
            }catch(InputMismatchException e)
            {
                System.out.println("Invalid selection, please input an integer number.");
                validSelection=false;
                sc.next();
            }
            catch(Exception e)
            {
                System.out.println("Something is error.");
            }
            }while(selection!=6 || !validSelection);
    }

    public void deleteProduct()
    {
```

```
int selection = 0;
do
{
	String productCode;
	boolean validSelection = false;
	try //catch mismatch input error and other possible errors
	{
	System.out.println("Enter your selection.");
	menus.menu11(); //ask user to choose delete record or leave
	selection = sc.nextInt();
	while(selection!=1 && selection!=2) //check if selection in range
	{
		System.out.println("Invalid selection, please input either 1 or 2.");
		System.out.println("Enter your selection.");
		menus.menu11();
		selection = sc.nextInt();
	}
	switch(selection)
	{
	case 1: //(1) Delete Records
		boolean validProductCode = true;
		int x = 0, z=1; //to check whether need to consume new line
		do
		{
			try //catch mismatch input error and other possible errors
			{
		int sel=0, del=0; //sel is selection whether want to delete, del is the index of the product need to delete
		System.out.println("Enter the product code of the record that you want to remove.");
		System.out.print("Product Code: ");
```

```
                    if(validProductCode|| ((x!=0)&&(validProductCode))||
(!validProductCode && z==0))
                    {
                        sc.nextLine(); //Consume new line
                    }
                    productCode = sc.nextLine().toUpperCase();
                    for (Product product_ : products)
                {
                  if(product_.getProductCode().equals(productCode)) //find the record that
need to be delete
                  {
                     System.out.println(productCode+" is found.");
                     validProductCode = true;
                     System.out.println("Are you sure you want to delete it permanently?");
                     menus.menu10(); //ask user to choose yes or no
                     sel = sc.nextInt();
                     if(sel==1) //(1) Yes, which is delete record
                     {
                         validSelection = true;
                        del = products.indexOf(product_); //get the index of product that
need to delete
                        z++;
                     }
                     else if (sel==2)
                     {
                         validSelection = true;
                         x=0;
                         System.out.println(productCode+" has not deleted.");
                         x++; /*variable x is used to determine whether need to print the
line for consume new line
                         the reason is it can check whether the code goes through this
line*/
```

```
                        z++;
                        break;
                    }
                    else {
                        {
                            z=0;
                            validSelection = false;
                            System.out.println("Invalid selection. Please input either 1
or 2.");
                        }
                                }
                }
                else
                {
                     validSelection = false;
                     validProductCode = false; //to make the consume new line
condition work
                }
            }
                        if(sel==1) //if user choose to delete
                        {
                            products.remove(del); //delete the product
                            System.out.println(productCode+" successfully deleted.");
                            break;
                        }
                        if(!validProductCode)
        {
            System.out.println("The product code does not exist in the system.");
        }
                            }catch(InputMismatchException e)
                            {
```

```
                                        System.out.println("Invalid selection, please input
an integer number.");
                                        validSelection=false;
                                        sc.next();
                                }
                                catch(Exception e)
                                {
                                        System.out.println("Something is error.");
                                }
                        }while(!validProductCode || !validSelection); /*will keep looping if
either invalid productCode
                        or invalid selection*/
                        break;

                case 2: // (2) No, which is cancel deletion
                        break;
                }
                }catch(InputMismatchException e)
                {
                        System.out.println("Invalid selection, please input an integer
number.");
                        validSelection=false;
                        sc.next();
                }
                catch(Exception e)
                {
                        System.out.println("Something is error.");
                }
        }while(selection!=2);
    }
```

```
    public void displayProduct()
    {
            int displaySelection=0, countrySelection, processorSelection, hardDiskSelection,
            internalMemorySelection, x, y, z;
            boolean validSelection = false, validSelection2 = false;
            do
            {
            try { //catch mismatch input error and other possible errors
menus.menu3(); //display display menu
displaySelection = sc.nextInt();
while(displaySelection < 1 || displaySelection > 7) //validate selection in range
{
   System.out.println("Invalid selection. Please input between 1 and 7.");
   menus.menu3(); //display display menu
   displaySelection = sc.nextInt();
}
switch(displaySelection)
{
   case 1 :  //(1) Display According to Manufacturing Country
            do
            {
            try //catch input mismatch error and other possible errors
            {
            validSelection=true;
            int sortingSelection=0;
            boolean validSortingSelection = false;
            System.out.println("Select the Manufacturing Country.");
            menus.menu4(); //display country menu
      countrySelection = sc.nextInt();
      while(countrySelection < 1 || countrySelection > 3) //validate selection in range
```

```
            {
               System.out.println("Invalid selection. Please input between 1 and 3.");
               System.out.println("Select the Manufacturing Country.");
               menus.menu4(); //display country menu
               countrySelection = sc.nextInt();
            }
            do
            {
            try
            {
            System.out.println("Select the criteria to sorted as."); //Sort according to criteria
            System.out.println("(1) Processor Type");
            System.out.println("(2) Hard Disk Capacity");
            System.out.print("Selection: ");
            sortingSelection = sc.nextInt();
            while (sortingSelection!= 1 && sortingSelection!=2) //validate sorting selection in
range
            {
                  System.out.println("Invalid selection. Please input either 1 or 2.");
                  System.out.println("Select the criteria to sorted as.");
               System.out.println("(1) Processor Type");
               System.out.println("(2) Hard Disk Capacity");
               System.out.print("Selection: ");
               sortingSelection = sc.nextInt();
            }
            products.sort(new Comparator<Product>()  //sort the list for binary search
                  {
               public int compare(Product p1, Product p2) {
                  return p1.getCountry().compareTo(p2.getCountry());
               }
```

```
            });
            validSortingSelection = true;
            }catch(InputMismatchException e)
            {
                System.out.println("Invalid selection, please input an integer number.");
                        validSortingSelection = false;
                        sc.next();
            }
            catch(Exception e)
            {
                System.out.println("Something wrong.");
            }
            }while(!validSortingSelection);
            switch(countrySelection)
            {
               case 1: //(1) Malaysia
                    ArrayList<Product> productsClone = new ArrayList<>(products); // declare clone array for binary search
                    Collections.copy(productsClone,products); //copy product array to a clone for binary search
                    ArrayList<Product> productsDisplay1 = new ArrayList<>();
                    validSelection2 = true;
                    int Index1 = 0;
                    Product targetProduct1 = new Product("", "Malaysia", "", "", "", 0);
                    while(Index1>=0) //loop until no result as binary search can only return one result one time
                    {
                    // Binary search for the first occurrence of the target product
                    if(!productsClone.isEmpty())
                    {
```

```
                Index1 = Collections.binarySearch(productsClone, targetProduct1, new
Comparator<Product>()
                {
                   public int compare(Product p1, Product p2)
                   {
                      return p1.getCountry().compareTo(p2.getCountry());
                   }
                });
                }
                else
                Index1=-1;
                if(Index1>=0) //if entry found, add the entry to the display array and remove from
clone array
                {
                        productsDisplay1.add(productsClone.get(Index1));
                        productsClone.remove(Index1);
                }
                }
                // Only print headers if there are matching records
                if (!productsDisplay1.isEmpty())
                {
                   menus.menu(); //print header
                   switch(sortingSelection) //sort according user's desires
                   {
                    case 1: //sort according to processor
                          productsDisplay1.sort(new Comparator<Product>()
                          {
                           public int compare(Product p1, Product p2) {
                               return p1.getProcessor().compareTo(p2.getProcessor());
                           }
                       });
```

```
            for (Product product : productsDisplay1) //display results
            {
                System.out.print(product);
            }
            break;
        case 2: //sort according to hard disk capacity
            productsDisplay1.sort(new Comparator<Product>() {
                public int compare(Product p1, Product p2) {
                    String hdd1 = p1.getHardDiskCapacity();
                    String hdd2 = p2.getHardDiskCapacity();

                    if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                        return 0;
                    } else if (hdd1.equals("320 GB")) { // 320 GB
                        return -1;
                    } else if (hdd2.equals("320 GB")) { // 320 GB
                        return 1;
                    } else if (hdd1.equals("500 GB")) { // 500 GB
                        return -1;
                    } else if (hdd2.equals("500 GB")) { // 500 GB
                        return 1;
                    } else if (hdd1.equals("1 TB")) {  // 1 TB
                        return -1;
                    } else {
                        return 1;                                                    // 1 TB
                    }
                }
            });
                    for (Product product : productsDisplay1) //display results
                    {
```

```
                                        System.out.print(product);
                    }
                    break;
                }
            }
            // Print "no records" message if no records were found
            else
            {
                System.out.println("No record with manufacturing country of Malaysia.");
            }
            break;


        case 2: //(2) Japan
            ArrayList<Product> productsClone2 = new ArrayList<>(products); // declare clone array for binary search
            Collections.copy(productsClone2,products); //copy product array to a clone for binary search
            ArrayList<Product> productsDisplay2 = new ArrayList<>();
            validSelection2 = true;
            int Index2 = 0;
            Product targetProduct2 = new Product("", "Japan", "", "", "", 0);
            while(Index2>=0) //loop until no result as binary search can only return one result one time
            {
            // Binary search for the first occurrence of the target product
            if(!productsClone2.isEmpty())
            {
            Index2 = Collections.binarySearch(productsClone2, targetProduct2, new Comparator<Product>()
            {
                public int compare(Product p1, Product p2)
```

```
                    {
                        return p1.getCountry().compareTo(p2.getCountry());
                    }
                });
                }
                else
                Index2=-1;
                if(Index2>=0) //if entry found, add the entry to the display array and remove from clone array
                {
                        productsDisplay2.add(productsClone2.get(Index2));
                        productsClone2.remove(Index2);
                }
                }
                // Only print headers if there are matching records
                if (!productsDisplay2.isEmpty())
                {
                    menus.menu(); //print header
                    switch(sortingSelection) //sort according user's desires
                    {
                    case 1: //sort according to processor
                            productsDisplay2.sort(new Comparator<Product>()
                            {
                             public int compare(Product p1, Product p2) {
                                return p1.getProcessor().compareTo(p2.getProcessor());
                             }
                        });
                            for (Product product : productsDisplay2) //display results
                            {
                                System.out.print(product);
```

```
            }
            break;
        case 2: //sort according to hard disk capacity
            productsDisplay2.sort(new Comparator<Product>() {
                public int compare(Product p1, Product p2) {
                    String hdd1 = p1.getHardDiskCapacity();
                    String hdd2 = p2.getHardDiskCapacity();

                    if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                        return 0;
                    } else if (hdd1.equals("320 GB")) { // 320 GB
                        return -1;
                    } else if (hdd2.equals("320 GB")) { // 320 GB
                        return 1;
                    } else if (hdd1.equals("500 GB")) { // 500 GB
                        return -1;
                    } else if (hdd2.equals("500 GB")) { // 500 GB
                        return 1;
                    } else if (hdd1.equals("1 TB")) {  // 1 TB
                        return -1;
                    } else {
                        return 1;                                          // 1 TB
                    }
                }
            });
                    for (Product product : productsDisplay2) //display results
                    {
                                    System.out.print(product);
                    }
                    break;
```

```
                    }
                }
                // Print "no records" message if no records were found
                else
                {
                    System.out.println("No record with manufacturing country of Japan.");
                }
                break;


            case 3: //(3) America
                ArrayList<Product> productsClone3 = new ArrayList<>(products); // declare clone array for binary search
                Collections.copy(productsClone3,products); //copy product array to a clone for binary search
                ArrayList<Product> productsDisplay3 = new ArrayList<>();
                validSelection2 = true;
                int Index3 = 0;
                Product targetProduct3 = new Product("", "America", "", "", "", 0);
                while(Index3>=0) //loop until no result as binary search can only return one result one time
                {
                // Binary search for the first occurrence of the target product
                if(!productsClone3.isEmpty())
                {
                Index3 = Collections.binarySearch(productsClone3, targetProduct3, new Comparator<Product>()
                {
                    public int compare(Product p1, Product p2)
                    {
                        return p1.getCountry().compareTo(p2.getCountry());
                    }
```

```
                });
                }
                else
                Index3=-1;
                if(Index3>=0) //if entry found, add the entry to the display array and remove from
clone array
                {
                        productsDisplay3.add(productsClone3.get(Index3));
                        productsClone3.remove(Index3);
                }
                }
                // Only print headers if there are matching records
                if (!productsDisplay3.isEmpty())
                {
                   menus.menu(); //print header
                   switch(sortingSelection) //sort according user's desires
                   {
                   case 1: //sort according to processor
                        productsDisplay3.sort(new Comparator<Product>()
                        {
                         public int compare(Product p1, Product p2) {
                            return p1.getProcessor().compareTo(p2.getProcessor());
                         }
                       });
                        for (Product product : productsDisplay3) //display results
                        {
                            System.out.print(product);
                        }
                        break;
                   case 2: //sort according to hard disk capacity
```

```
                productsDisplay3.sort(new Comparator<Product>() {
                    public int compare(Product p1, Product p2) {
                        String hdd1 = p1.getHardDiskCapacity();
                        String hdd2 = p2.getHardDiskCapacity();

                        if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                            return 0;
                        } else if (hdd1.equals("320 GB")) { // 320 GB
                            return -1;
                        } else if (hdd2.equals("320 GB")) { // 320 GB
                            return 1;
                        } else if (hdd1.equals("500 GB")) { // 500 GB
                            return -1;
                        } else if (hdd2.equals("500 GB")) { // 500 GB
                            return 1;
                        } else if (hdd1.equals("1 TB")) {  // 1 TB
                            return -1;
                        } else {
                            return 1;                                                    // 1 TB
                        }
                    }
                });
                        for (Product product : productsDisplay3) //display results
                        {
                                        System.out.print(product);
                        }
                        break;
        }
    }
    // Print "no records" message if no records were found
```

```
                else
                {
                    System.out.println("No record with manufacturing country of America.");
                }
                break;

            default:
                System.out.println("Invalid selection. Please try again.");
                System.exit(countrySelection);
        }
                }catch(InputMismatchException e)
                {
                        System.out.println("Invalid selection, please input an integer number.");
                        validSelection2=false;
                        sc.next();
                }
                catch(Exception e)
                {
                        System.out.println("Something is error.");
                        System.out.println(e);
                }
                }while(!validSelection2);
            break;

        case 2: //(2) Display According to Processor Type
                do
                {
                try //catch input mismatch error and other possible errors
                {
                validSelection=true;
```

```
            x=0; //to check if got product equals to i5
            y=0; //to check if got product equals to i7
            z=0; //to check if got product equals to i9
            int sortingSelection;
            boolean validSortingSelection = false;
            System.out.println("Select the Processor Type.");
        menus.menu5();//display processor type menu
        processorSelection = sc.nextInt();
        while(processorSelection < 1 || processorSelection > 3) //validate selection in range
        {
           System.out.println("Invalid selection. Please input between 1 and 3.");
                        System.out.println("Select the Processor Type.");
           menus.menu5(); //display processor type menu
           processorSelection = sc.nextInt();
        }
        do
        {
        try
        {
        System.out.println("Select the criteria to sorted as."); //Sort according to criteria
        System.out.println("(1) Manufacturing Country");
        System.out.println("(2) Hard Disk Capacity");
        System.out.print("Selection: ");
        sortingSelection = sc.nextInt();
        while (sortingSelection!= 1 && sortingSelection!=2) //validate sorting selection in
range
        {
            System.out.println("Invalid selection. Please input either 1 or 2.");
            System.out.println("Select the criteria to sorted as.");
            System.out.println("(1) Manufacturing Country");
```

```
            System.out.println("(2) Hard Disk Capacity");
            System.out.print("Selection: ");
            sortingSelection = sc.nextInt();
        }
        switch(sortingSelection)
        {
        case 1: //sort according country
                validSortingSelection = true;
                products.sort(new Comparator<Product>()
                {
                 public int compare(Product p1, Product p2) {
                    return p1.getCountry().compareTo(p2.getCountry());
                }
            });
                break;
        case 2: //sort according hard disk capacity
                validSortingSelection = true;
                products.sort(new Comparator<Product>() {
                    public int compare(Product p1, Product p2) {
                        String hdd1 = p1.getHardDiskCapacity();
                        String hdd2 = p2.getHardDiskCapacity();

                        if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                            return 0;
                        } else if (hdd1.equals("320 GB")) { // 320 GB
                            return -1;
                        } else if (hdd2.equals("320 GB")) { // 320 GB
                            return 1;
                        } else if (hdd1.equals("500 GB")) { // 500 GB
                            return -1;
```

```
                } else if (hdd2.equals("500 GB")) { // 500 GB
                    return 1;
                } else if (hdd1.equals("1 TB")) {  // 1 TB
                    return -1;
                } else {
                    return 1;                                                    // 1 TB
                }
            }
        });
        break;
    }
    }catch(InputMismatchException e)
    {
        System.out.println("Invalid selection, please input an integer number.");
                validSortingSelection = false;
                sc.next();
    }
    catch(Exception e)
    {
        System.out.println("Something wrong.");
    }
    }while(!validSortingSelection);
    switch(processorSelection)
    {
       case 1: //(1) Intel i5
          validSelection2=true;
          System.out.println("The product records with processor type of Intel i5");
          for(Product product : products)
          {
             if(product.getProcessor().equals("Intel i5"))
```

```
                    {
                        x++;
                    }
                }
                if (x==0)
                {
                        System.out.println("No record with processor type of Intel i5.");
                }
                else if (x!=0)
                {
                        menus.menu(); //print headers
                        for (Product product : products)
                        {
                                if(product.getProcessor().equals("Intel i5")) //print records that
processor i5
                    {
                        System.out.print(product);
                    }
                        }
                }
                break;

            case 2: //(2) Intel i7
                validSelection2=true;
                System.out.println("The product records with processor type of Intel i7");
                for(Product product : products)
                {
                    if(product.getProcessor().equals("Intel i7"))
                    {
                        y++;
```

```
            }
        }
        if (y==0)
        {
                System.out.println("No record with processor type of Intel i7.");
        }
        else if (y!=0)
        {
                menus.menu(); //print headers
                for (Product product : products) //print records that processor i7
                {
                        if(product.getProcessor().equals("Intel i7"))
            {
                 System.out.print(product);
            }
                  }
        }
        break;

    case 3: //(3) Intel i9
       validSelection2=true;
       System.out.println("The product records with processor type of Intel i9");
       for(Product product : products)
       {
          if(product.getProcessor().equals("Intel i9"))
          {
             z++;
          }
       }
       if (z==0)
```

```
                {
                        System.out.println("No record with processor type of Intel i9.");
                }
                else if (z!=0)
                {
                        menus.menu(); //print headers
                        for (Product product : products) //print records with processor i9
                        {
                                if(product.getProcessor().equals("Intel i9"))
                       {
                         System.out.print(product);
                       }
                         }
                }
                break;

            default:
               System.out.println("Invalid selection. Please try again.");
               System.exit(processorSelection);
        }
               }catch(InputMismatchException e)
               {
                        System.out.println("Invalid selection, please input an integer number.");
                        validSelection2=false;
                        sc.next();
               }
               catch(Exception e)
               {
                        System.out.println("Something is error.");
               }
```

```
            }while(!validSelection2);
        break;


    case 3 : //(3) Display According to Hard Disk Capacity
            do
            {
            try //catch input mismatch error and other possible errors
            {
            validSelection=true;
            int sortingSelection = 1;
            boolean validSortingSelection = false;
            System.out.println("Select the Hard Disk Capacity.");
        menus.menu6(); //display hard disk menu
        hardDiskSelection = sc.nextInt();
        while(hardDiskSelection < 1 || hardDiskSelection > 3) //validate selection range
        {
            System.out.println("Invalid selection. Please input between 1 and 3.");
                        System.out.println("Select the Hard Disk Capacity.");
            menus.menu6(); //display hard disk menu
            hardDiskSelection = sc.nextInt();
        }
        do
        {
        try
        {
        System.out.println("Select the criteria to sorted as."); //Sort according to criteria
        System.out.println("(1) Manufacturing Country");
        System.out.println("(2) Processor Type");
        System.out.print("Selection: ");
        sortingSelection = sc.nextInt();
```

```
        while (sortingSelection!= 1 && sortingSelection!=2) //validate sorting selection in
range
        {
            System.out.println("Invalid selection. Please input either 1 or 2.");
            System.out.println("Select the criteria to sorted as.");
            System.out.println("(1) Manufacturing Country");
            System.out.println("(2) Processor Type");
           System.out.print("Selection: ");
           sortingSelection = sc.nextInt();
        }
        products.sort(new Comparator<Product>() //sort the list for binary search
            {
           public int compare(Product p1, Product p2) {
              return p1.getHardDiskCapacity().compareTo(p2.getHardDiskCapacity());
           }
        });
        validSortingSelection = true;
        }catch(InputMismatchException e)
        {
            System.out.println("Invalid selection, please input an integer number.");
                    validSortingSelection = false;
                    sc.next();
        }
        catch(Exception e)
        {
            System.out.println("Something wrong.");
        }
        }while(!validSortingSelection);
        switch(hardDiskSelection)
        {
```

```
            case 1: //(1) 320 GB
                ArrayList<Product> productsClone = new ArrayList<>(products); // declare clone array for binary search
                Collections.copy(productsClone,products); //copy product array to a clone for binary search
                ArrayList<Product> productsDisplay1 = new ArrayList<>();
                validSelection2 = true;
                int Index1 = 0;
                Product targetProduct1 = new Product("", "", "", "320 GB", "", 0);
                while(Index1>=0) //loop until no result as binary search can only return one result one time
                {
                // Binary search for the first occurrence of the target product
                if(!productsClone.isEmpty())
                {
                Index1 = Collections.binarySearch(productsClone, targetProduct1, new Comparator<Product>()
                {
                   public int compare(Product p1, Product p2)
                   {
                      return p1.getHardDiskCapacity().compareTo(p2.getHardDiskCapacity());
                   }
                });
                }
                else
                Index1=-1;
                if(Index1>=0) //if entry found, add the entry to the display array and remove from clone array
                {
                        productsDisplay1.add(productsClone.get(Index1));
                        productsClone.remove(Index1);
                }
```

```
}
// Only print headers if there are matching records
if (!productsDisplay1.isEmpty())
{
    menus.menu(); //print header
    switch(sortingSelection)
    {
    case 1:
            productsDisplay1.sort(new Comparator<Product>()
        {
            public int compare(Product p1, Product p2) {
                return p1.getCountry().compareTo(p2.getCountry());
            }
        });
          for (Product product : productsDisplay1)
          {
              System.out.print(product);
          }
          break;
    case 2:
          productsDisplay1.sort(new Comparator<Product>()
       {
           public int compare(Product p1, Product p2) {
               return p1.getProcessor().compareTo(p2.getProcessor());
           }
       });
          for (Product product : productsDisplay1)
          {
              System.out.print(product);
          }
```

```
                    break;
                }
            }
            // Print "no records" message if no records were found
            else
            {
                System.out.println("No record with hard disk capacity of 320 GB.");
            }
            break;

        case 2 : //(2) 500 GB
            ArrayList<Product> productsClone2 = new ArrayList<>(products); // declare clone array for binary search
            Collections.copy(productsClone2,products); //copy product array to a clone for binary search
            ArrayList<Product> productsDisplay2 = new ArrayList<>();
            validSelection2 = true;
            int Index2 = 0;
            Product targetProduct2 = new Product("", "", "", "500 GB", "", 0);
            while(Index2>=0) //loop until no result as binary search can only return one result one time
            {
            // Binary search for the first occurrence of the target product
            if(!productsClone2.isEmpty())
            {
            Index2 = Collections.binarySearch(productsClone2, targetProduct2, new Comparator<Product>()
            {
                public int compare(Product p1, Product p2)
                {
                    return p1.getHardDiskCapacity().compareTo(p2.getHardDiskCapacity());
```

```
                }
            });
            }
            else
            Index2=-1;
            if(Index2>=0) //if entry found, add the entry to the display array and remove from
clone array
            {
                    productsDisplay2.add(productsClone2.get(Index2));
                    productsClone2.remove(Index2);
            }
            }
            // Only print headers if there are matching records
            if (!productsDisplay2.isEmpty())
            {
               menus.menu(); //print header
               switch(sortingSelection)
               {
               case 1:
                     productsDisplay2.sort(new Comparator<Product>()
                  {
                     public int compare(Product p1, Product p2) {
                        return p1.getCountry().compareTo(p2.getCountry());
                     }
                  });
                    for (Product product : productsDisplay2)
                    {
                        System.out.print(product);
                    }
                    break;
```

```
                    case 2:
                        productsDisplay2.sort(new Comparator<Product>()
                    {
                        public int compare(Product p1, Product p2) {
                            return p1.getProcessor().compareTo(p2.getProcessor());
                        }
                    });
                        for (Product product : productsDisplay2)
                        {
                            System.out.print(product);
                        }
                        break;
                    }
                }
                // Print "no records" message if no records were found
                else
                {
                    System.out.println("No record with hard disk capacity of 500 GB.");
                }
                break;


            case 3 : //(3) 1 TB
                ArrayList<Product> productsClone3 = new ArrayList<>(products); // declare clone
array for binary search
                Collections.copy(productsClone3,products); //copy product array to a clone for
binary search
                ArrayList<Product> productsDisplay3 = new ArrayList<>();
                validSelection2 = true;
                int Index3 = 0;
                Product targetProduct3 = new Product("", "", "", "1 TB", "", 0);
```

```
            while(Index3>=0) //loop until no result as binary search can only return one result
one time
            {
            // Binary search for the first occurrence of the target product
            if(!productsClone3.isEmpty())
            {
            Index3 = Collections.binarySearch(productsClone3, targetProduct3, new
Comparator<Product>()
            {
               public int compare(Product p1, Product p2)
               {
                  return p1.getHardDiskCapacity().compareTo(p2.getHardDiskCapacity());
               }
            });
            }
            else
            Index3=-1;
            if(Index3>=0) //if entry found, add the entry to the display array and remove from
clone array
            {
                  productsDisplay3.add(productsClone3.get(Index3));
                  productsClone3.remove(Index3);
            }
            }
            // Only print headers if there are matching records
            if (!productsDisplay3.isEmpty())
            {
               menus.menu(); //print header
               switch(sortingSelection)
               {
               case 1:
```

```
                productsDisplay3.sort(new Comparator<Product>()
            {
                public int compare(Product p1, Product p2) {
                    return p1.getCountry().compareTo(p2.getCountry());
                }
            });
                for (Product product : productsDisplay3)
                {
                    System.out.print(product);
                }
                break;
        case 2:
                productsDisplay3.sort(new Comparator<Product>()
            {
                public int compare(Product p1, Product p2) {
                    return p1.getProcessor().compareTo(p2.getProcessor());
                }
            });
                for (Product product : productsDisplay3)
                {
                    System.out.print(product);
                }
                break;
        }
    }
    // Print "no records" message if no records were found
    else
    {
        System.out.println("No record with hard disk capacity of 1 TB.");
    }
```

```
                break;


            default:
                System.out.println("Invalid selection. Please try again.");
                System.exit(hardDiskSelection);
        }
                }catch(InputMismatchException e)
                {
                        System.out.println("Invalid selection, please input an integer number.");
                        validSelection2=false;
                        sc.next();
                }
                catch(Exception e)
                {
                        System.out.println("Something is error.");
                }
                }while(!validSelection2);
            break;

        case 4: //(4) Display According to Internal Memory Capacity
                do
                {
                try //catch input mismatch error and other possible errors
                {
                validSelection=true;
                int sortingSelection = 1;
                boolean validSortingSelection = false;
                System.out.println("Select the Internal Memory Capacity.");
                menus.menu7(); //display internal memory capacity menu
            internalMemorySelection = sc.nextInt();
```

```
        while(internalMemorySelection < 1 || internalMemorySelection > 3) //validate selection
in range
        {
           System.out.println("Invalid selection. Please input between 1 and 3.");
                        System.out.println("Select the Internal Memory Capacity.");
           menus.menu7(); //display internal memory capacity menu
           internalMemorySelection = sc.nextInt();
        }
        do
        {
        try
        {
        System.out.println("Select the criteria to sorted as."); //Sort according to criteria
        System.out.println("(1) Manufacturing Country");
        System.out.println("(2) Processor Type");
        System.out.println("(3) Hard Disk Capacity");
        System.out.print("Selection: ");
        sortingSelection = sc.nextInt();
        while (sortingSelection<1 || sortingSelection>3) //validate sorting selection in range
        {
              System.out.println("Invalid selection. Please input between 1 and 3.");
              System.out.println("Select the criteria to sorted as.");
              System.out.println("(1) Manufacturing Country");
           System.out.println("(2) Processor Type");
           System.out.println("(3) Hard Disk Capacity");
           System.out.print("Selection: ");
           sortingSelection = sc.nextInt();
        }
        products.sort(new Comparator<Product>() //sort the list for binary search
              {
```

```
                public int compare(Product p1, Product p2) {
                    return
p1.getInternalMemoryCapacity().compareTo(p2.getInternalMemoryCapacity());
                }
            });
            validSortingSelection = true;
            }catch(InputMismatchException e)
            {
                System.out.println("Invalid selection, please input an integer number.");
                        validSortingSelection = false;
                        sc.next();
            }
            catch(Exception e)
            {
                System.out.println("Something wrong.");
            }
            }while(!validSortingSelection);
            switch(internalMemorySelection){
               case 1: //(1) 1 TB
                  ArrayList<Product> productsClone = new ArrayList<>(products); // declare clone
array for binary search
                  Collections.copy(productsClone,products); //copy product array to a clone for
binary search
                  ArrayList<Product> productsDisplay1 = new ArrayList<>();
                  validSelection2 = true;
                  int Index1 = 0;
                  Product targetProduct1 = new Product("", "", "", "", "1 TB", 0);
                  while(Index1>=0) //loop until no result as binary search can only return one result
one time
                  {
                  // Binary search for the first occurrence of the target product
```

```
                if(!productsClone.isEmpty())
                {
                Index1 = Collections.binarySearch(productsClone, targetProduct1, new
Comparator<Product>()
                {
                   public int compare(Product p1, Product p2)
                   {
                      return
p1.getInternalMemoryCapacity().compareTo(p2.getInternalMemoryCapacity());
                   }
                });
                }
                else
                Index1=-1;
                if(Index1>=0) //if entry found, add the entry to the display array and remove from
clone array
                {
                        productsDisplay1.add(productsClone.get(Index1));
                        productsClone.remove(Index1);
                }
                }
                // Only print headers if there are matching records
                if (!productsDisplay1.isEmpty())
                {
                   menus.menu(); //print header
                   switch(sortingSelection)
                   {
                   case 1:
                       productsDisplay1.sort(new Comparator<Product>()
                       {
                          public int compare(Product p1, Product p2) {
```

```
                return p1.getCountry().compareTo(p2.getCountry());
            }
        });
        for (Product product : productsDisplay1)
        {
            System.out.print(product);
        }
        break;

    case 2:
        productsDisplay1.sort(new Comparator<Product>()
        {
            public int compare(Product p1, Product p2) {
                return p1.getProcessor().compareTo(p2.getProcessor());
            }
        });
        for (Product product : productsDisplay1)
        {
            System.out.print(product);
        }
        break;

    case 3:
        productsDisplay1.sort(new Comparator<Product>() {
         public int compare(Product p1, Product p2) {
            String hdd1 = p1.getHardDiskCapacity();
            String hdd2 = p2.getHardDiskCapacity();

            if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                return 0;
```

```
                    } else if (hdd1.equals("320 GB")) { // 320 GB
                        return -1;
                    } else if (hdd2.equals("320 GB")) { // 320 GB
                        return 1;
                    } else if (hdd1.equals("500 GB")) { // 500 GB
                        return -1;
                    } else if (hdd2.equals("500 GB")) { // 500 GB
                        return 1;
                    } else if (hdd1.equals("1 TB")) {    // 1 TB
                        return -1;
                    } else {
                        return 1;                       // 1 TB
                    }
                }
            });
            for (Product product : productsDisplay1)
              {
                  System.out.print(product);
              }
              break;
        }
    }
    // Print "no records" message if no records were found
    else
    {
        System.out.println("No record with internal memory capacity of 1 TB.");
    }
    break;

case 2: //(2) 2 GB
```

```
ArrayList<Product> productsClone2 = new ArrayList<>(products); // declare clone array for binary search

Collections.copy(productsClone2,products); //copy product array to a clone for binary search

ArrayList<Product> productsDisplay2 = new ArrayList<>();

validSelection2 = true;

int Index2 = 0;

Product targetProduct2 = new Product("", "", "", "", "2 GB", 0);

while(Index2>=0) //loop until no result as binary search can only return one result one time

{

// Binary search for the first occurrence of the target product

if(!productsClone2.isEmpty())

{

Index2 = Collections.binarySearch(productsClone2, targetProduct2, new Comparator<Product>()

{

   public int compare(Product p1, Product p2)

   {

      return p1.getInternalMemoryCapacity().compareTo(p2.getInternalMemoryCapacity());

   }

});

}

else

Index2=-1;

if(Index2>=0) //if entry found, add the entry to the display array and remove from clone array

{

        productsDisplay2.add(productsClone2.get(Index2));

        productsClone2.remove(Index2);

}
```

```
}
// Only print headers if there are matching records
if (!productsDisplay2.isEmpty())
{
    menus.menu(); //print header
    switch(sortingSelection)
    {
    case 1:
        productsDisplay2.sort(new Comparator<Product>()
        {
            public int compare(Product p1, Product p2) {
                return p1.getCountry().compareTo(p2.getCountry());
            }
        });
          for (Product product : productsDisplay2)
          {
              System.out.print(product);
          }
          break;

    case 2:
          productsDisplay2.sort(new Comparator<Product>()
         {
             public int compare(Product p1, Product p2) {
                 return p1.getProcessor().compareTo(p2.getProcessor());
             }
        });
          for (Product product : productsDisplay2)
          {
              System.out.print(product);
```

```
            }
            break;

        case 3:
            productsDisplay2.sort(new Comparator<Product>() {
             public int compare(Product p1, Product p2) {
                String hdd1 = p1.getHardDiskCapacity();
                String hdd2 = p2.getHardDiskCapacity();

                if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                    return 0;
                } else if (hdd1.equals("320 GB")) { // 320 GB
                    return -1;
                } else if (hdd2.equals("320 GB")) { // 320 GB
                    return 1;
                } else if (hdd1.equals("500 GB")) { // 500 GB
                    return -1;
                } else if (hdd2.equals("500 GB")) { // 500 GB
                    return 1;
                } else if (hdd1.equals("1 TB")) {    // 1 TB
                    return -1;
                } else {
                    return 1;                       // 1 TB
                }
             }
            });
            for (Product product : productsDisplay2)
            {
                System.out.print(product);
            }
```

```
                    break;
                }
            }
            // Print "no records" message if no records were found
            else
            {
                System.out.println("No record with internal memory capacity of 2 GB.");
            }
            break;

        case 3: //(3) 4 GB
            ArrayList<Product> productsClone3 = new ArrayList<>(products); // declare clone array for binary search
            Collections.copy(productsClone3,products); //copy product array to a clone for binary search
            ArrayList<Product> productsDisplay3 = new ArrayList<>();
            validSelection2 = true;
            int Index3 = 0;
            Product targetProduct3 = new Product("", "", "", "", "4 GB", 0);
            while(Index3>=0) //loop until no result as binary search can only return one result one time
            {
            // Binary search for the first occurrence of the target product
            if(!productsClone3.isEmpty())
            {
            Index3 = Collections.binarySearch(productsClone3, targetProduct3, new Comparator<Product>()
            {
                public int compare(Product p1, Product p2)
                {
                    return p1.getInternalMemoryCapacity().compareTo(p2.getInternalMemoryCapacity());
```

```
                    }
                });
                }
                else
                Index3=-1;
                if(Index3>=0) //if entry found, add the entry to the display array and remove from
clone array
                {
                        productsDisplay3.add(productsClone3.get(Index3));
                        productsClone3.remove(Index3);
                }
                }
                // Only print headers if there are matching records
                if (!productsDisplay3.isEmpty())
                {
                   menus.menu(); //print header
                   switch(sortingSelection)
                   {
                   case 1:
                       productsDisplay3.sort(new Comparator<Product>()
                       {
                          public int compare(Product p1, Product p2) {
                              return p1.getCountry().compareTo(p2.getCountry());
                          }
                       });
                         for (Product product : productsDisplay3)
                         {
                             System.out.print(product);
                         }
                         break;
```

```
case 2:
       productsDisplay3.sort(new Comparator<Product>()
     {
         public int compare(Product p1, Product p2) {
            return p1.getProcessor().compareTo(p2.getProcessor());
         }
   });
       for (Product product : productsDisplay3)
       {
           System.out.print(product);
       }
       break;

case 3:
       productsDisplay3.sort(new Comparator<Product>() {
        public int compare(Product p1, Product p2) {
           String hdd1 = p1.getHardDiskCapacity();
           String hdd2 = p2.getHardDiskCapacity();

           if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
              return 0;
           } else if (hdd1.equals("320 GB")) { // 320 GB
              return -1;
           } else if (hdd2.equals("320 GB")) { // 320 GB
              return 1;
           } else if (hdd1.equals("500 GB")) { // 500 GB
              return -1;
           } else if (hdd2.equals("500 GB")) { // 500 GB
              return 1;
```

```
                    } else if (hdd1.equals("1 TB")) {    // 1 TB
                        return -1;
                    } else {
                        return 1;                        // 1 TB
                    }
                }
            });
            for (Product product : productsDisplay3)
              {
                  System.out.print(product);
              }
              break;
        }
    }
    // Print "no records" message if no records were found
    else
    {
       System.out.println("No record with internal memory capacity of 1 TB.");
    }
    break;

  default:
    System.out.println("Invalid selection. Please try again.");
     System.exit(internalMemorySelection);
}
    }catch(InputMismatchException e)
    {
            System.out.println("Invalid selection, please input an integer number.");
            validSelection2=false;
            sc.next();
```

```
                }
                catch(Exception e)
                {
                        System.out.println("Something is error.");
                }
                }while(!validSelection2);
            break;

        case 5: //(5) Display According to Quantity
                boolean validQuantity = false;
                do
                {
                try //catch input mismatch error and other possible erros
                {
                x=0; //to check if there is product that quantity same with input quantity
                validSelection=true;
                int quant, sortingSelection;
                boolean validSortingSelection = false;
                System.out.println("Please enter the number of quantity.");
                System.out.print("Quantity: ");
                quant = sc.nextInt();
                while(quant < 0) //validate quantity should be positive
                {
                        System.out.println("Invalid number of quantity. Please enter an positive
integer.");
                        System.out.println("Please enter the number of quantity.");
                        System.out.print("Quantity: ");
                quant = sc.nextInt();
                }
                do
```

```
{
try
{
System.out.println("Select the criteria to sorted as."); //Sort according to criteria
System.out.println("(1) Manufacturing Country");
System.out.println("(2) Processor Type");
System.out.println("(3) Hard Disk Capacity");
System.out.print("Selection: ");
sortingSelection = sc.nextInt();
while (sortingSelection<1 || sortingSelection>3) //validate sorting selection in range
{
       System.out.println("Invalid selection. Please input between 1 and 3.");
       System.out.println("Select the criteria to sorted as.");
       System.out.println("(1) Manufacturing Country");
   System.out.println("(2) Processor Type");
   System.out.println("(3) Hard Disk Capacity");
   System.out.print("Selection: ");
   sortingSelection = sc.nextInt();
}
switch(sortingSelection)
{
case 1:
       validSortingSelection = true;
       products.sort(new Comparator<Product>()
       {
        public int compare(Product p1, Product p2) {
           return p1.getCountry().compareTo(p2.getCountry());
       }
   });
       break;
```

```
            case 2:
                validSortingSelection = true;
                products.sort(new Comparator<Product>()
                {
                 public int compare(Product p1, Product p2) {
                    return p1.getProcessor().compareTo(p2.getProcessor());
                }
               });
                break;
            case 3:
                validSortingSelection = true;
                products.sort(new Comparator<Product>() {
                   public int compare(Product p1, Product p2) {
                      String hdd1 = p1.getHardDiskCapacity();
                      String hdd2 = p2.getHardDiskCapacity();

                      if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                         return 0;
                      } else if (hdd1.equals("320 GB")) { // 320 GB
                         return -1;
                      } else if (hdd2.equals("320 GB")) { // 320 GB
                         return 1;
                      } else if (hdd1.equals("500 GB")) { // 500 GB
                         return -1;
                      } else if (hdd2.equals("500 GB")) { // 500 GB
                         return 1;
                      } else if (hdd1.equals("1 TB")) {  // 1 TB
                         return -1;
                      } else {
                         return 1;                                          // 1 TB
```

```
                }
            }
        });
        break;
    }
    }catch(InputMismatchException e)
    {
        System.out.println("Invalid selection, please input an integer number.");
                validSortingSelection = false;
                sc.next();
    }
    catch(Exception e)
    {
        System.out.println("Something wrong.");
    }
    }while(!validSortingSelection);
        validQuantity = true;
        for(Product product : products)
    {
       if(product.getQuantity()==(quant))
       {
          x++;
       }
    }
        if (x==0)
        {
                System.out.println("No record with quanity of "+quant+".");
        }
        else if (x!=0)
    {
```

```
            menus.menu(); //print headers
            for (Product product : products) //print products with selected quantity
            {
                    if(product.getQuantity()==(quant))
            {
               System.out.print(product);
            }
            }
        }
            }catch(InputMismatchException e)
            {
                    System.out.println("Invalid quantity, please input an integer number.");
                    validQuantity=false;
                    sc.next();
            }
            catch(Exception e)
            {
                    System.out.println("Something is error.");
            }
            }while(!validQuantity);
        break;

    case 6: //(6) Display All
            int sortingSelection = 1;
            boolean validSortingSelection = false;
            validSelection = true;
            do
        {
        try
        {
```

```
System.out.println("Select the criteria to sorted as."); //Sort according to criteria
System.out.println("(1) Manufacturing Country");
System.out.println("(2) Processor Type");
System.out.println("(3) Hard Disk Capacity");
System.out.print("Selection: ");
sortingSelection = sc.nextInt();
while (sortingSelection<1 || sortingSelection>3) //validate sorting selection in range
{
      System.out.println("Invalid selection. Please input between 1 and 3.");
      System.out.println("Select the criteria to sorted as.");
      System.out.println("(1) Manufacturing Country");
   System.out.println("(2) Processor Type");
   System.out.println("(3) Hard Disk Capacity");
   System.out.print("Selection: ");
   sortingSelection = sc.nextInt();
}
switch(sortingSelection)
{
case 1:
      validSortingSelection = true;
      products.sort(new Comparator<Product>()
      {
       public int compare(Product p1, Product p2) {
          return p1.getCountry().compareTo(p2.getCountry());
       }
   });
      break;
case 2:
      validSortingSelection = true;
      products.sort(new Comparator<Product>()
```

```
                {
                 public int compare(Product p1, Product p2) {
                     return p1.getProcessor().compareTo(p2.getProcessor());
                 }
               });
                 break;
            case 3:
                 validSortingSelection = true;
                 products.sort(new Comparator<Product>() {
                    public int compare(Product p1, Product p2) {
                       String hdd1 = p1.getHardDiskCapacity();
                       String hdd2 = p2.getHardDiskCapacity();

                       if (hdd1 == hdd2) { //sort by 320GB, 500GB, 1TB
                          return 0;
                       } else if (hdd1.equals("320 GB")) { // 320 GB
                          return -1;
                       } else if (hdd2.equals("320 GB")) { // 320 GB
                          return 1;
                       } else if (hdd1.equals("500 GB")) { // 500 GB
                          return -1;
                       } else if (hdd2.equals("500 GB")) { // 500 GB
                          return 1;
                       } else if (hdd1.equals("1 TB")) {  // 1 TB
                          return -1;
                       } else {
                          return 1;                                          // 1 TB
                       }
                    }
                 });
```

```
                break;
            }
                menus.menu(); //print headers
                for (Product product : products) //print all products
                {
                 System.out.print(product);
                }
            }catch(InputMismatchException e)
            {
                System.out.println("Invalid selection, please input an integer number.");
                        validSortingSelection = false;
                        sc.next();
            }
            catch(Exception e)
            {
                System.out.println("Something wrong.");
            }
            }while(!validSortingSelection);
        case 7: //(7) Exit
                validSelection = true;
                break;
    }
                }catch(InputMismatchException e)
                {
                        System.out.println("Invalid selection, please input an integer number.");
                        validSelection=false;
                        sc.next();
                }
                catch(Exception e)
                {
```

```
                        System.out.println("Something is error.");

                }

        }while(displaySelection!=7 || !validSelection); //will keep looping if user not choose to exit or
invalid selection

        }

}
```

1.1.3 Menu.java

```
package Assignment2;

public class Menu {

	public void menu() //Header

	{


	System.out.println("=========================================================
"

				+
"=================================================================");

		System.out.println("Product Code \tCountry \tProcessor\tHard Disk Capacity\t"

				+ "Internal Memory Capacity\tQuantity");


	System.out.println("=========================================================
"

				+
"=================================================================");

	}

	public void menu1() //Product code table

	{

		System.out.println("\t\t    Product Code Table");


	System.out.println("=========================================================
==========");

		System.out.println("\tMeaning \t      Characters \t Translation");


	System.out.println("=========================================================
==========");

		System.out.println("Manufacturing Country \t\t  M \t\t Malaysia");

		System.out.println("\t\t\t\t  J \t\t Japan");

		System.out.println("\t\t\t\t  A \t\t America");


	System.out.println("=========================================================
==========");

		System.out.println("Type of Processor \t\t  5 \t\t Intel i5");
```

```
                System.out.println("\t\t\t\t  7 \t\t Intel i7");
                System.out.println("\t\t\t\t  9 \t\t Intel i9");

        System.out.println("==========================================================
==========");
                System.out.println("Capacity of Hard Disk \t\t  320 \t\t 320 GB");
                System.out.println("\t\t\t\t  500 \t\t 500 GB");
                System.out.println("\t\t\t\t  1024 \t\t 1 TB");

        System.out.println("==========================================================
==========");
                System.out.println("Capacity of Internal Memory \t  1024 \t\t 1 TB");
                System.out.println("\t\t\t\t  2048 \t\t 2 GB");
                System.out.println("\t\t\t\t  4096 \t\t 4 GB");

        System.out.println("==========================================================
==========");
                System.out.println("The product code table is only for references before any
updation or deletion.\n");
        }
        public void menu2() //Main Menu
        {
                System.out.println("Please key in your selection.");
                System.out.println("(1) Add New Products");
                System.out.println("(2) Update Records");
      System.out.println("(3) Delete Records");
      System.out.println("(4) Display Records Based on Different Criteria");
      System.out.println("(5) Display Product Code Table");
      System.out.println("(6) Exit ");
      System.out.print("Selection: ");
        }
        public void menu3() //(4) Display Records Based on Different Criteria
```

```
    {
System.out.println("Select the criteria you wish to view products based on it.");
System.out.println("(1) Display According to Manufacturing Country");
System.out.println("(2) Display According to Processor Type");
System.out.println("(3) Display According to Hard Disk Capacity");
System.out.println("(4) Display According to Internal Memory Capacity");
System.out.println("(5) Display According to Quantity");
System.out.println("(6) Display All");
System.out.println("(7) Exit");
System.out.print("Selection: ");
    }
    public void menu4() //(1)  Manufacturing Country
    {
System.out.println("(1) Malaysia");
System.out.println("(2) Japan");
System.out.println("(3) America");
System.out.print("Selection: ");
    }
    public void menu5() //(2) Processor Type
    {
System.out.println("(1) Intel i5");
System.out.println("(2) Intel i7");
System.out.println("(3) Intel i9");
System.out.print("Selection: ");
    }
    public void menu6() //(3) Hard Disk Capacity
    {
            System.out.println("(1) 320 GB");
System.out.println("(2) 500 GB");
System.out.println("(3) 1 TB");
```

```
System.out.print("Selection: ");
   }
   public void menu7() //(4) Internal Memory Capacity
   {
System.out.println("(1) 1 TB ");
System.out.println("(2) 2 GB ");
System.out.println("(3) 4 GB ");
System.out.print("Selection: ");
   }
   public void menu8() //(2) Update Records
   {
            System.out.println("Select the criteria of the product you wish to update.");
System.out.println("(1) Manufacturing Country");
System.out.println("(2) Processor Type");
System.out.println("(3) Hard Disk Capacity");
System.out.println("(4) Internal Memory Capacity");
System.out.println("(5) Quantity");
System.out.println("(6) Exit");
System.out.print("Selection: ");
   }
   public void menu9() //Update all or update one by one
   {
            System.out.println("(1) Update All");
            System.out.println("(2) Update One");
            System.out.print("Selection: ");
   }
   public void menu10() //Yes or no menu
   {
            System.out.println("(1) Yes");
            System.out.println("(2) No");
```

```
            System.out.print("Selection: ");
        }
        public void menu11() //(3) Delete Records
        {
            System.out.println("(1) Delete Records");
            System.out.println("(2) Exit");
            System.out.print("Selection: ");
        }
        public void menu12() // (1) Add New Products
        {
            System.out.println("(1) Add Records");
            System.out.println("(2) Exit");
            System.out.print("Selection: ");
        }
}
```

1.1.4 Product.java

```
package Assignment2;


public class Product {
        //instance variables
        private String productCode;  //the code that identifies the product
        private String country; //the country where the product is manufactured
    private String processor; //the type of processor used in the product
    private String hardDiskCapacity; // the capacity of the hard disk in the product
    private String internalMemoryCapacity; //the capacity of the internal memory in the product
    private int quantity; //the quantity of the product available


    // Constructor
    public Product(String productCode,String country, String processor,
                    String hardDiskCapacity, String internalMemoryCapacity,int quantity)
    {
        this.productCode = productCode;
            this.country = country;
        this.processor = processor;
        this.hardDiskCapacity = hardDiskCapacity;
        this.internalMemoryCapacity = internalMemoryCapacity;
        this.quantity = quantity;
    }
    //Empty constructor for creating an object with default values
    public Product()
    {


    }
    //Getters and Setters for the instance variables
    public String getProductCode()
```

```
    {
        return productCode;
    }
    public void setProductCode(String productCode)
    {
        this.productCode = productCode;
    }
    public String getCountry()
    {
       return country;
    }

    public void setCountry(String country)
    {
       this.country = country;
    }

    public String getProcessor()
    {
       return processor;
    }

    public void setProcessor(String processor)
    {
       this.processor = processor;
    }

    public String getHardDiskCapacity()
    {
       return hardDiskCapacity;
```

```
    }

    public void setHardDiskCapacity(String hardDiskCapacity)
    {
        this.hardDiskCapacity = hardDiskCapacity;
    }

    public String getInternalMemoryCapacity()
    {
        return internalMemoryCapacity;
    }

    public void setInternalMemoryCapacity(String internalMemoryCapacity)
    {
        this.internalMemoryCapacity = internalMemoryCapacity;
    }

    public int getQuantity()
    {
        return quantity;
    }

    public void setQuantity(int quantity)
    {
        this.quantity = quantity;
    }
    // Returns a formatted string representation of the object
    public String toString()
    {
```

```
        return String.format("%-16s%-17s%-20s%-28s%-26s%d\n", productCode,country,
processor,
                        hardDiskCapacity, internalMemoryCapacity, quantity);
    }


}
```

1.2 Documentation

1.2.1 Demonstration of Enhanced Features

Since most of the program will work exactly same as the documentation in Assignment 1, the documentation in this Assignment 2 will be only demonstrating the new features which is sorting features (only for manufacturing country, processor type and hard disk capacity) and searching by using binary search (only for manufacturing country, hard disk capacity and internal memory capacity) in displaying product records.

After running the program and select desired view records in display records function, the system will prompt the sorting selection for the user to choose the desired sorting criteria for the records. For example, it will allow user to sort as processor type or hard disk capacity for display according to manufacturing country; to sort as manufacturing country or hard disk capacity for display according to processor type; to sort as manufacturing country or processor type for display according to hard disk capacity; to sort as manufacturing country, processor type or hard disk capacity for display according to internal memory capacity, display according to quantity and display all as shown in *Figure 1.1*, *Figure 1.2*, *Figure 1.3*, *Figure 1.4*, *Figure 1.5*, *Figure 1.6*.

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 1
Select the Manufacturing Country.
(1) Malaysia
(2) Japan
(3) America
Selection: 1
Select the criteria to sorted as.
(1) Processor Type
(2) Hard Disk Capacity
Selection:
```

Figure 1.1 Sorting Selection – Display According to Manufacturing Country

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 2
Select the Processor Type.
(1) Intel i5
(2) Intel i7
(3) Intel i9
Selection: 1
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Hard Disk Capacity
Selection:
```

Figure 1.2 Sorting Selection – Display According to Processor Type

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 3
Select the Hard Disk Capacity.
(1) 320 GB
(2) 500 GB
(3) 1 TB
Selection: 1
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
Selection:
```

Figure 1.3 Sorting Selection – Display According to Hard Disk Capacity

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 4
Select the Internal Memory Capacity.
(1) 1 TB
(2) 2 GB
(3) 4 GB
Selection: 1
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection:
```

Figure 1.4 Sorting Selection – Display According to Internal Memory Capacity

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 5
Please enter the number of quantity.
Quantity: 10
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection:
```

Figure 1.5 Sorting Selection – Display According to Quantity

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 6
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection:
```

Figure 1.6 Sorting Selection – Display All

After user selected the desired sorting criteria, the system will display the user's desired record according to their selected sorting criteria. For example, user wish to view all products in the system and wish to sort them according to the manufacturing country. The system displays all product records and sorts them by country name in alphabetically, which is America, Japan and Malaysia as shown in *Figure 1.7*.

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 6
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection: 1
|=============================================================================================================|
Product Code    Country         Processor       Hard Disk Capacity      Internal Memory Capacity        Quantity
=============================================================================================================
A53201024       America         Intel i5        320 GB                  1 TB                    20
A510244096      America         Intel i5        1 TB                    4 GB                    20
A910241024      America         Intel i9        1 TB                    1 TB                    10
J53201024       Japan           Intel i5        320 GB                  1 TB                    5
J95002048       Japan           Intel i9        500 GB                  2 GB                    5
J510242048      Japan           Intel i5        1 TB                    2 GB                    5
M73201024       Malaysia        Intel i7        320 GB                  1 TB                    10
M55002048       Malaysia        Intel i5        500 GB                  2 GB                    10
M910244096      Malaysia        Intel i9        1 TB                    4 GB                    20
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection:
```

Figure 1.7 Output – Display All (Sort by Manufacturing Country)

Another example, user wish to view Malaysia records in the system and wish to sort them according to the processor type. The system displays Malaysia product records and sorts them by processor type, which is Intel i5, Intel i7 and Intel i9 as shown in *Figure 1.8*.

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 1
Select the Manufacturing Country.
(1) Malaysia
(2) Japan
(3) America
Selection: 1
Select the criteria to sorted as.
(1) Processor Type
(2) Hard Disk Capacity
Selection: 1
=============================================================================================================
Product Code    Country         Processor       Hard Disk Capacity      Internal Memory Capacity        Quantity
=============================================================================================================
M55002048       Malaysia         Intel i5           500 GB                    2 GB                      10
M73201024       Malaysia         Intel i7           320 GB                    1 TB                      10
M910244096      Malaysia         Intel i9           1 TB                      4 GB                      20
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection:
```

Figure 1.8 Output – Display Malaysia Records (Sort by Processor Type)

Moreover, if user wish to view Intel i5 records in the system and wish to sort them according to the hard disk capacity, the system displays Intel i5 product records and sorts them by hard disk capacity, which is 320 GB, 500 GB and 1 TB as shown in *Figure 1.9*.

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 2
Select the Processor Type.
(1) Intel i5
(2) Intel i7
(3) Intel i9
Selection: 1
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Hard Disk Capacity
Selection: 2
The product records with processor type of Intel i5
=============================================================================================================
Product Code    Country         Processor       Hard Disk Capacity      Internal Memory Capacity        Quantity
=============================================================================================================
A53201024       America          Intel i5           320 GB                    1 TB                      20
J53201024       Japan            Intel i5           320 GB                    1 TB                      5
M55002048       Malaysia         Intel i5           500 GB                    2 GB                      10
A510244096      America          Intel i5           1 TB                      4 GB                      20
J510242048      Japan            Intel i5           1 TB                      2 GB                      5
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection:
```

Figure 1.9 Output – Display Intel i5 Records (Sort by Hard Disk Capacity)

1.2.2 Validation of Enhanced Features

The validation will only be showing the validation on new features as well as mentioned in 1.2.1 Demonstration of Enhanced Features.

If the user input integer that out of sorting selection range, the system will prompt the user to key in a valid sorting selection as shown in *Figure 2.1*, *Figure 2.2*.

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 1
Select the Manufacturing Country.
(1) Malaysia
(2) Japan
(3) America
Selection: 1
Select the criteria to sorted as.
(1) Processor Type
(2) Hard Disk Capacity
Selection: 0
Invalid selection. Please input either 1 or 2.
Select the criteria to sorted as.
(1) Processor Type
(2) Hard Disk Capacity
Selection:
```

Figure 2.1 Sorting Selection Out of Range – 2 Option

```
Select the criteria you wish to view products based on it.
(1) Display According to Manufacturing Country
(2) Display According to Processor Type
(3) Display According to Hard Disk Capacity
(4) Display According to Internal Memory Capacity
(5) Display According to Quantity
(6) Display All
(7) Exit
Selection: 4
Select the Internal Memory Capacity.
(1) 1 TB
(2) 2 GB
(3) 4 GB
Selection: 1
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection: 4
Invalid selection. Please input between 1 and 3.
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection:
```

Figure 2.2 Sorting Selection Out of Range – 3 Option

If the user key in invalid sorting selection that is not an integer, the system will prompt the user to key in an valid integer sorting selection as shown in *Figure 2.3*.

```
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection: a
Invalid selection, please input an integer number.
Select the criteria to sorted as.
(1) Manufacturing Country
(2) Processor Type
(3) Hard Disk Capacity
Selection:
```

Figure 2.3 Sorting Selection Not Numeric

2.0 Task B - Individual Report [Chan Seow Fen (0207368)]

Search algorithms are used to perform checking or retrieving an element from any data structure that contains that element. They look in the search space for a target (key) (Krishna, 2022). One of common searching algorithms is linear search. Linear search is also known as sequential search, it is a simple and straightforward search method that systematically examines each element of an array or list, beginning with the first element, until the desired element is located. Although it is the simplest and most straightforward search technique, it is not the most effective for huge datasets. The linear search algorithm works by start with the array's or list's first element. Next, up until the target element is located or the array's end is reached, compare each element with the target element. Finally, return the element's index if the desired element can be located. If not, return -1. For a better illustration, in real life, suppose Ali have a bookcase full of titles and Ali is trying to locate a certain title. To find the book Ali is looking for, start with the first book on the left and scan each one. Similar to the linear search searching algorithm, this procedure involves systematically scanning each element of an array or list until he locates the desired element. As an illustration, suppose Ali is looking for "Data Structures and Algorithms" on his bookshelf, which also contains the following titles: Foundation of HCI, Object Oriented Programming, Data Structures and Algorithms, System Analysis and Design, Computer Networks. If Ali wanted to find " Data Structures and Algorithms " using linear search, Ali would start from "Foundation of HCI," the first book on the left, and scan each book until he found it. To locate the target book in this example, scan the second book ("Object Oriented Programming") first. Although this procedure is straightforward and easy to understand, it gets less effective as the bookshelf fills up with more titles. Other real-life examples for linear search are finding a name on a phonebook, finding a person without knowing his or her appearance in a queue and finding a specific document in an unsorted document shelf. The time complexity of linear search is O(1), when the target element is discovered at the start of the array or list, which is the best case situation for linear search. In contrast, the worst-case situation involves finding the target element at the end of the array or list, which results in an O(n) time complexity. The target element is equally likely to be present at any place in the array or list in the average case scenario, resulting in an average time complexity of O(n) (S, 2021). The space complexity of linear search algorithm is O(1) which means it has a constant memory need (Ue Kiao, PhD, 2021). The algorithm just utilizes a few variables to hold the index and target element, and it does not need to store any other data structures. As a result, the magnitude of the input data has no bearing on how complex the linear search's space is.

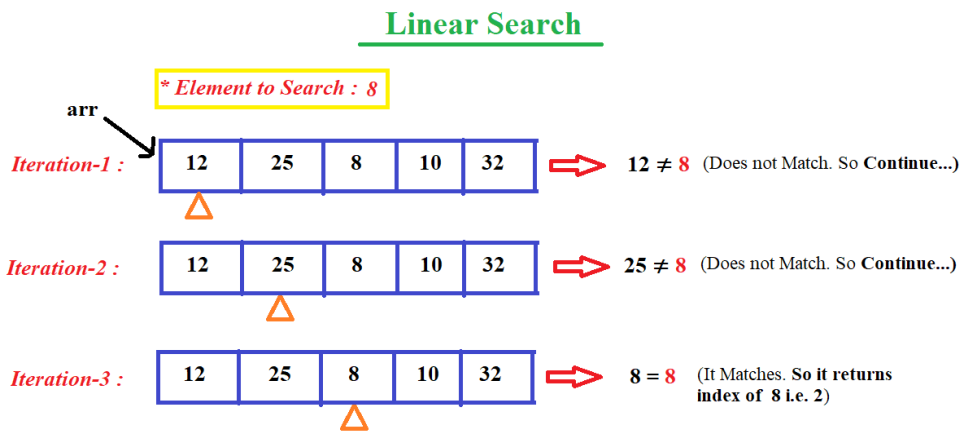

Figure 3.1 Linear Search Algorithm (Github.io, 2023)

Another searching algorithm is binary search. For sorted arrays, binary search is a more effective searching algorithm than linear search. The search period is split in half repeatedly until the target element is located or the search interval is empty. The binary search algorithm work by first, determine the array's middle index. Then, return the target element's index if the middle element is identical to it. It will repeat the method on the right half of the array if the center element is less than the target element and discard the left half. It will repeat the method on the left half of the array if the middle element is greater than the target element and discard the right half. It will continue doing this until the search interval is empty or the target element is located. To illustrate it, consider a library with a big collection of books where Amin is trying to find a specific book. All of the books in the library are ordered alphabetically by last name of the author. Amin is trying to locate the book as soon as he can, and he is aware of the author's last name. In this case, binary search might be used to find the book. He would begin by scanning the middle of the library's book selection. He would then search the right half of the collection if the book he is looking for appears after the book in the middle of the collection alphabetically. Amin would search the left half of the collection if the book came before the one in the middle. The search space would be divided in half again and again until the desired book was located. Because the books are grouped alphabetically, which is a sorted dataset, binary search is especially helpful in this situation. Amin may rapidly find the book he is seeking for without having to browse through every book in the library by using binary search. Overall, binary search can be applied to a variety of real-world situations where it is necessary to search a sorted dataset in order to locate a specific item, such as looking up a certain name in a phone book or a specific item in the inventory of an online retailer. The time complexity of binary search in the best case scenario will be O(1) if the target element being located in the center of the sorted array or list as binary search start from the middle of an array. However, in the worst situation, the target element is not present in the array or list, leading to an O(log n) time complexity as binary search need to search over the array until it make sure the element does not exist. Nevertheless, the target element is equally likely to be present in any place in the sorted array or list which is the average case scenario, leading to an average time complexity of O(log n) (Jana, 2022). The space complexity of binary search will be also O(1) as well. The reason is that binary search only requires a constant amount of memory to execute as to keep track of the variety of components that must be examined, only two variables are required. No other information is required. However, that is only applicable for common use of binary search which is iterative implementation of binary search, if it is recursive implementation of binary search, the space complexity will be O(log n) as in the worst situation, log n recursive calls will occur and that they will all be piled in memory. In fact, if I comparisons are required, I recursive calls will be stacked in memory, and based on our analysis of the average case time complexity, it can infer that the average memory will also be O(log n) (Ue Kiao, PhD, 2021).

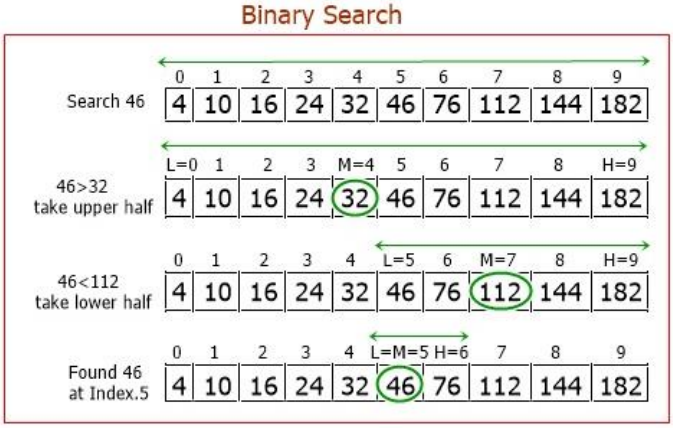

Figure 3.2 Binary Search Algorithm (Codescracker.com, 2023)

3.0 Reference List

Codescracker.com. (2023). *Binary Search Logic and Algorithm with Example*. [online] Available at: https://codescracker.com/computer-fundamental/binary-search.htm [Accessed 19 Apr. 2023].

Github.io. (2023). *Linear Search – Searching & Sorting – Data Structures & Algorithms*. [online] Available at: https://utkarsh1504.github.io/DSA-Java/linearsearch [Accessed 18 Apr. 2023].

Jana, S. (2022). *What is the Time Complexity of Binary Search? - Scaler Topics*. [online] Scaler Topics. Available at: https://www.scaler.com/topics/time-complexity-of-binary-search/ [Accessed 19 Apr. 2023].

Krishna, A. (2022). *Search Algorithms – Linear Search and Binary Search Code Implementation and Complexity Analysis*. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/search-algorithms-linear-and-binary-search-explained/ [Accessed 18 Apr. 2023].

S, R.A. (2021). *Linear Search Algorithm: Overview, Complexity, Implementation*. [online] Simplilearn.com. Available at: https://www.simplilearn.com/tutorials/data-structure-tutorial/linear-search-algorithm [Accessed 19 Apr. 2023].

Ue Kiao, PhD (2021). *Time & Space Complexity of Binary Search [Mathematical Analysis]*. [online] OpenGenus IQ: Computing Expertise & Legacy. Available at: https://iq.opengenus.org/time-complexity-of-binary-search/ [Accessed 19 Apr. 2023].

Ue Kiao, PhD (2021). *Time & Space Complexity of Linear Search [Mathematical Analysis]*. [online] OpenGenus IQ: Computing Expertise & Legacy. Available at: https://iq.opengenus.org/time-complexity-of-linear-search/ [Accessed 19 Apr. 2023].