-----------------------------------------------------------------------------------------------------------------------------------------

***Declaration of understanding***

*I understand that plagiarised answers will receive a grade of zero and an possible academic misconduct.*

*I understand that an academic misconduct will result in an automatic zero in the exam in the event that this is my first offense and an automatic grade of zero in the course if this is my second offense.*

*I understand that all answer material, including sample code, must be original material.*

*I understand that “less is more”, meaning that it is better to write less material(which is original) than to write more of someone else’s material, in this case the less will score more grades.*

*I understand that plagiarism means to present someone else’s material as my own.*

*I understand that to copy material from the internet, text books, course slides, or any other source, is a form of plagiarism.*

*I understand that to copy material from the internet, text books, course slides or any other source, and then to change a few words in that material is a form of plagiarism.*

*I understand that to copy material from the internet, text books, course slides or any other source, and then to paraphrase that material is a form of plagiarism.*

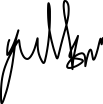
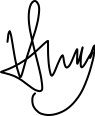
*I understand that the appropriate way to answer these exam questions is to write the answers in essay form or bullet point form in my own words based upon my current understanding of the material without referencing any material at this present time.*

*I understand that an oral one-on-one solution review will be requested if the submitted answers are materially beyond the caliber of work that I have demonstrated to date, and that I will receive a grade of zero if I am unable to clearly explain my answers to the satisfaction of the course Professor.*

*I have read and clearly understand each one of these statements, and accept the responsibility and penalty for any actions that I take which may contravene any one of these statements.*

*Name (type)\_\_\_\_\_\_\_\_Yu-Hsuan Huang\_\_\_\_\_\_\_\_*

*Student Number: \_\_\_\_\_\_\_\_\_\_\_\_\_200443723\_\_\_\_\_\_\_\_\_\_\_\_\_*



*Signature: (draw with mouse or tablet pen):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

**Composition:**

1. Composition is a part of an object. Using “has a” relationship. And this relationship is one-way. It cannot reverse this relationship. In addition these object also can individually exist in different class.
2. Create Payment class that will be included in Invoice class.

Invoice

Payment

Information

behavior

Information

behavior

has a

**public** **class** Payment {

**private** String paymentType;

**private** **int** money;

//Constructor

**public** Payment() {}

//Getter and Setter

**public** String getPaymentType() {

**return** paymentType;

}

**public** **void** setPaymentType(String paymentType) {

**this**.paymentType = paymentType;

}

**public** **int** getMoney() {

**return** money;

}

**public** **void** setMoney(**int** money) {

**this**.money = money;

}

}

1. The Payment class is used to indicate the payment type and payment amount.
2. Create Invoice class which will include Payment object.

**public** **class** Invoice {

**private** Payment payment;

**private** String title;

//Constructor

**public** Invoice() {}

//Getter and Setter

**public** Payment getPayment() {

**return** payment;

}

**public** **void** setPayment(Payment payment) {

**this**.payment = payment;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

}

1. This Invoice class has a payment object. This is a composition concept which a class has other object which include information and behavior.

**Re-use:**

1. Re-use means class can be re-use for instantiate many objects. Thus, we don't need to create class for each entity.
2. Create Invoice class which will be instantiate many object.

Class

Invoice1

Invoice

instantiate

Invoice2

**public** **class** Invoice {

**private** Payment payment;

**private** String title;

//Constructor

**public** Invoice() {}

//Getter and Setter

**public** Payment getPayment() {

**return** payment;

}

**public** **void** setPayment(Payment payment) {

**this**.payment = payment;

}

**public** String getTitle() {

**return** title;

}

**public** **void** setTitle(String title) {

**this**.title = title;

}

}

1. This invoice class will be re-use for instantiate many object.
2. Create test class and main function use to instantiate invoice objects.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Invoice invoice1 = **new** Invoice();

Invoice invoice2 = **new** Invoice();

}

}

1. This is a re-use sample. We can instantiate 2 or more invoice objects. Like invoice1 object and invoice2 object.

**Constructors/constructor overloading:**

1. Constructor is a public method which uses to initialize objects and constructor must use public access level and no return type. Overloading means methods have same name, but different amount of arguments OR different type of arguments
2. Create Payment with default constructor and constructor which takes all arguments.

**public** **class** Payment {

**private** String paymentType;

**private** **int** money;

//Default constructor

**public** Payment() {}

//Constructor takes all arguments

**public** Payment(String paymentType, **int** money) {

**this**.paymentType = paymentType;

**this**.money = money;

}

//Constructor take string argument

**public** Payment(String paymentType) {

**this**.paymentType = paymentType;

}

//Constructor take int argument

**public** Payment(**int** money) {

**this**.money = money;

}

}

1. Payment class has two constructors. First is default constructor that don’t have any argument and if you don’t write any constructor, it will be automatically generated. Second constructor which has the same name takes two arguments.
2. Create test class to test each constructor

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Payment payment1 = **new** Payment();

Payment payment2 = **new** Payment("Bill", 200);

Payment payment3 = **new** Payment("Bill");

Payment payment4 = **new** Payment(200);

}

}

1. We can see payment1 object is instantiated with no argument constructor but payment2 is instantiated with 2 argument constructor. Java overloading will choose use which method based on number of argument and type of argument. Like payment3 object and payment4 object.

**Switch statement:**

1. Switch statement likes if statement but it is multi-way statement. The value use in statement only can be strings, integers and convertible integers. And we need to use break keyword to leave each case if not it will fall through to the next case.
2. Create switch statement sample

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int** option = 2;

**switch** (option) {

**case** 1:

System.***out***.println("Option equal to 1");

**break**;

**case** 2:

**case** 3:

System.***out***.println("Option equal to 2 or 3");

**break**;

**default**:

System.***out***.println("Option not equal from 1 to 3");

}

}

}

1. This is switch statement sample. If option value equal to 1, it will print “Option equal to 1” and leave this switch statement. If option value equal to 2 or 3, it will print “Option equal to 2 or 3”. The case 2 will fall through to case 3 because case 2 has no break keywork. If the option value not match any case, it will go inside default area.

**Static method:**

1. Static method can be used without instantiate the object and it only exists in the class. It mean it cannot be instantiated for each object. In addition, static method only can use static variable.
2. Create Payment class which include static method and static variable.

**public** **class** Payment {

**public** **static** **int** *serialNumber* = 1234;

**private** **int** money;

//Default constructor

**public** Payment() {}

**public** **static** **void** printSerialNumber() {

//correct

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

//error

//System.out.println(maney);

}

}

1. This payment class has static method and static variable. In printSerialNumber() method, it can use static variable but cannot use class variable.
2. Create Test class which will call this static method.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Payment.*printSerialNumber*();

}

}

1. We can call payment static method without instantiate an object.

**Multi-dimensional arrays:**

1. The multi-dimensional array is also an array, but it can be thought of as having another array under the array of the first layer. If it is a two-dimensional array, it can be thought a table. If it is a three-dimensional array, it can be thought a cube.
2. Instantiate two-dimensional string array in main function.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

//Instantiate two-dimensional string array

String[][] array = **new** String[10][20];

//Rows

**for**(**int** i=0; i<10; i++) {

//Columns

**for**(**int** j=0; j<20; j++) {

System.***out***.println("Row: " + i + "Column: " + j);

}

}

}

}

1. In this case, it will instantiate two-dimensional string array which has 10 rows and 20 columns. In the for loop, it will print row number and column number.

**Exception handling:**

1. Exception handling can assist us catch the error when the program running. When java through an exception we can get this information which help us to fixed this issue. It use to process the error that can be fixed or recovered.
2. Create a string array and try to use for loop iterate the array.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

//Instantiate string array

String[] strArray = **new** String[10];

**try** {

**for**(**int** i=0; i<11; i++) {

System.***out***.println("Index: " + i);

}

} **catch** (IndexOutOfBoundsException e) {

System.***out***.println("Index is out of Bounds");

}

}

}

4. In this for loop, the variable i will exceed the array size. It will throw this exception to catch area. After that, you can print this exception or fix this problem.

**Wrapper classes:**

1. Wrapper class is a class which can use primitive data type and can be instantiate to object. Wrapper class include many method which include convert others data type into this data type, compare data, get data length and so on.
2. Create a test class has primitive data type and wrapper class.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

//Primitive data

**int** integer1 = 1;

//Wrapper class

Integer integer2 = 2;

//Compare (equal)

integer2.equals(integer1);

//Convert

Integer.*valueOf*(integer1);

}

}

1. The variables integer1 and integer2 both are integer types but integer2 is a class that contains primitive data and specific method. And wrapper class can use some methods, such as equals() method, compare the data is the same or not and return a Boolean value.