Inheritance COW

Level 1

Create the following classes:

Shape	
protected double base, height	
Name:	Shape
Input:	double base, double height
Output:	nothing
Action:	constructor that takes in and initializes base and height
Name:	getArea
Input:	nothing
Output:	double area
Action:	calculates and returns the area. Use the formula for a rectangle

Triangle (extends the Shape class)	
No additional variables	
Name:	Triangle
Input:	double base, double height
Output:	nothing
Action:	constructor that takes in the base and height and passes them along to the constructor of the parent class by calling super
Name:	getArea
Input:	nothing
Output:	double area
Action:	calculates and returns the area. The getArea of the parent class is overridden and the formula for a triangle should be used

Trapezoid (extends the Shape class)	
protected double base2	
Name:	Trapezoid
Input:	double base1, double height, double base2
Output:	nothing
Action:	constructor that takes in the bases and height of a trapezoid. It passes base1 and height to the constructor of the parent class by calling super and initializes its own base2 variable
Name:	getArea
Input:	nothing
Output:	double area
Action:	calculates and returns the area. The getArea of the parent class is overridden and the formula for a trapezoid should be used

Level 2

Create the following classes:

Player

private String name;
private int number, gamesPlayed;

Name: Player
Input: String name, int number, int gamesPlayed
Output: nothing
Action: constructor that takes in and initializes variables

Name: toString
Input: nothing
Output: String description
Action: returns a description of the player in the form
"**name** – **number**\nGames Played: **gamesPlayed**"

BasketBallPlayer (extends the Player class)

protected int pointsScored, rebounds, steals, blocks;

Name: BasketBallPlayer
Input: String name, int number, int gamesPlayed, int pointsScored, int rebounds, int steals, int blocks
Output: nothing
Action: constructor that takes in and initializes variables. All appropriate variables should be passed to the constructor of the parent class.

Name: toString
Input: nothing
Output: String description
Action: returns a description of the player in the form
"**name** – **number**\nGames Played: **gamesPlayed**\nPoints Scored: **pointsScored**\nRebounds: **rebounds**\nSteals: **steals**\nBlocks: **blocks**"
The toString method of the parent class should be utilized.

FootballPlayer (extends the Player class)
protected int touchdowns, receptions, tackles;
Name: FootballPlayer Input: String name, int number, int gamesPlayed, int touchdowns, int receptions, int tackles Output: nothing Action: constructor that takes in and initializes variables. All appropriate variables should be passed to the constructor of the parent class. Name: toString Input: nothing Output: String description Action: returns a description of the player in the form "**name** – **number**\nGames Played: **gamesPlayed**\nTouchdowns: **touchdowns**\nReceptions: **receptions**\nTackles: **tackles**" The toString method of the parent class should be utilized.

Level 3

Create the following classes:

Dice
Name: roll Input: nothing Output: int result Action: returns a random number from 1 to 6. There should be an equal chance that each number is returned

MultiSidedDice (extends the Dice class)
private int numSides;
Name: MultiSidedDice Input: int numSides Output: nothing Action: constructor that takes in and initializes variables Name: roll Input: nothing Output: int result Action: returns a random number from 1 to numSides. There should be an equal chance that each number is returned

MultiValueDice (extends the Dice class)
protected int[] values;
Name: MultiValueDice Input: int[] values; Output: nothing Action: constructor that takes in and initializes variables Name: roll Input: nothing Output: int result Action: returns a random number stored in values. There should be an equal chance that each number is returned

Level 4

Create the following classes:

Function1	
private double h;	
Name:	Function1
Input:	double h;
Output:	nothing
Action:	constructor that takes in and initializes variables
Name:	getValueAt
Input:	int x
Output:	int result
Action:	returns the result of x-h.

Function2 (extends the Function1 class)	
private double e;	
Name:	Function2
Input:	double h, double e;
Output:	nothing
Action:	constructor that takes in and initializes variables. It should make use of calling the constructor of the parent class to set its variables.
Name:	getValueAt
Input:	int x
Output:	int result
Action:	returns the result of $(x\text{-}h)^e$. It should make use of getValueAt of the parent class.

Function3 (extends the Function2 class)	
private double a;	
Name:	Function3
Input:	double h, double e, double a
Output:	nothing
Action:	constructor that takes in and initializes variables. It should make use of calling the constructor of the parent class to set its variables.
Name:	getValueAt
Input:	int x
Output:	int result
Action:	returns the result of $a(x\text{-}h)^e$. It should make use of getValueAt of the parent class.

Function4 (extends the Function3 class)

private double k;

Name:	Function4
Input:	double h, double e, double a, double k
Output:	nothing
Action:	constructor that takes in and initializes variables. It should make use of calling the constructor of the parent class to set its variables.

Name:	getValueAt
Input:	int x
Output:	int result
Action:	returns the result of $a(x-h)^e+k$. It should make use of getValueAt of the parent class.

Adder

Name:	add
Input:	Object one, Object two
Output:	Object result
Action:	takes in two objects and returns a different object depending on the types of the two objects passed in. instanceof and casting Objects should be made use of to do this. The possible combinations are as follows:

- Two Characters – return a String that is made up of the two characters combined.
- Two Integers – returns an Integer that is the sum of the two Integers passed in
- Two Strings – returns a String that is made up of the two Strings passed in with a space in between
- Two Booleans – returns a Boolean that is a true if one of the Booleans passed in is true but the other is not. Two trues or two falsas result in a false being returned.

Level 5

Create the following classes:

Sorter
Name: sort Input: Person[] people Output: Person[] sortedPeople Action: returns the array passed in sorted using their names (alphabetical order). You can use any sorting algorithm you like.

AgeSorter (extends the Sorter class)
Name: sort Input: Person[] people Output: Person[] sortedPeople Action: returns the array passed in sorted using their age (low to high). You can use any sorting algorithm you like.

GenderSorter (extends the Sorter class)
Name: sort Input: Person[] people Output: Person[] sortedPeople Action: returns the array passed in sorted using their gender (male followed by female). You can use any sorting algorithm you like.

Level 6

Create the following classes:

MultiDice (extends the Dice class)
protected ArrayList<Dice> theDice;
Name: MultiDice Input: ArrayList<Dice> theDice; Output: nothing Action: constructor that takes in and initializes variables Name: roll Input: nothing Output: int result Action: returns the sum of the rolls of all the dice in the ArrayList theDice. There should be a bell curve distribution that is normally created when rolling multiple dice.

HighValueMultiDice (extends the MultiDice class)
Name: HighValueMultiDice Input: ArrayList<Dice> theDice; Output: nothing Action: constructor that takes in and initializes variables (calls super). Name: roll Input: nothing Output: int result Action: returns the highest roll of the rolls of the dice in the ArrayList theDice.

MultiDice (extends the MultiValueDice class)

protected double[] weights;
protected double totalWeight;

Name: MultiDice Input: int[] values, double[] weights Output: nothing Action: constructor that takes in and initializes variables. super should be made use of to set values. totalWeight should also be calculated that is the sum of all the weights in weights. Name: roll Input: nothing Output: int result Action: returns one of the ints stored in values. The chance that each value is returned should correspond to the corresponding weight stored in weights. For example, if there is a value 30 and it has a weight of 5 and the total of all the weights is 50, then there should be a 10% chance that it gets returned.

Update the following class (Adding people and Dice)

Adder
Name: add Input: Object one, Object two Output: Object result Action: takes in two objects and returns a different object depending on the types of the two objects passed in. instanceof and casting Objects should be made use of to do this. The possible combinations are as follows: • Two Characters – return a String that is made up of the two characters combined. • Two Integers – returns an Integer that is the sum of the two Integers passed in • Two Strings – returns a String that is made up of the two Strings passed in with a space in between • Two Booleans – returns a Boolean that is a true if one of the Booleans passed in is true but the other is not. Two trues or two falsas result in a false being returned. • Two Persons – returns a new Person with the name of the two Persons passed in combined with a hyphen in between, has an age of 0, and a random gender. • Two Dice – returns a MultiDice that consists of the two Dice passed in put in an ArrayList togethore