Tyler Freitas
07/16/2018
cs162_400

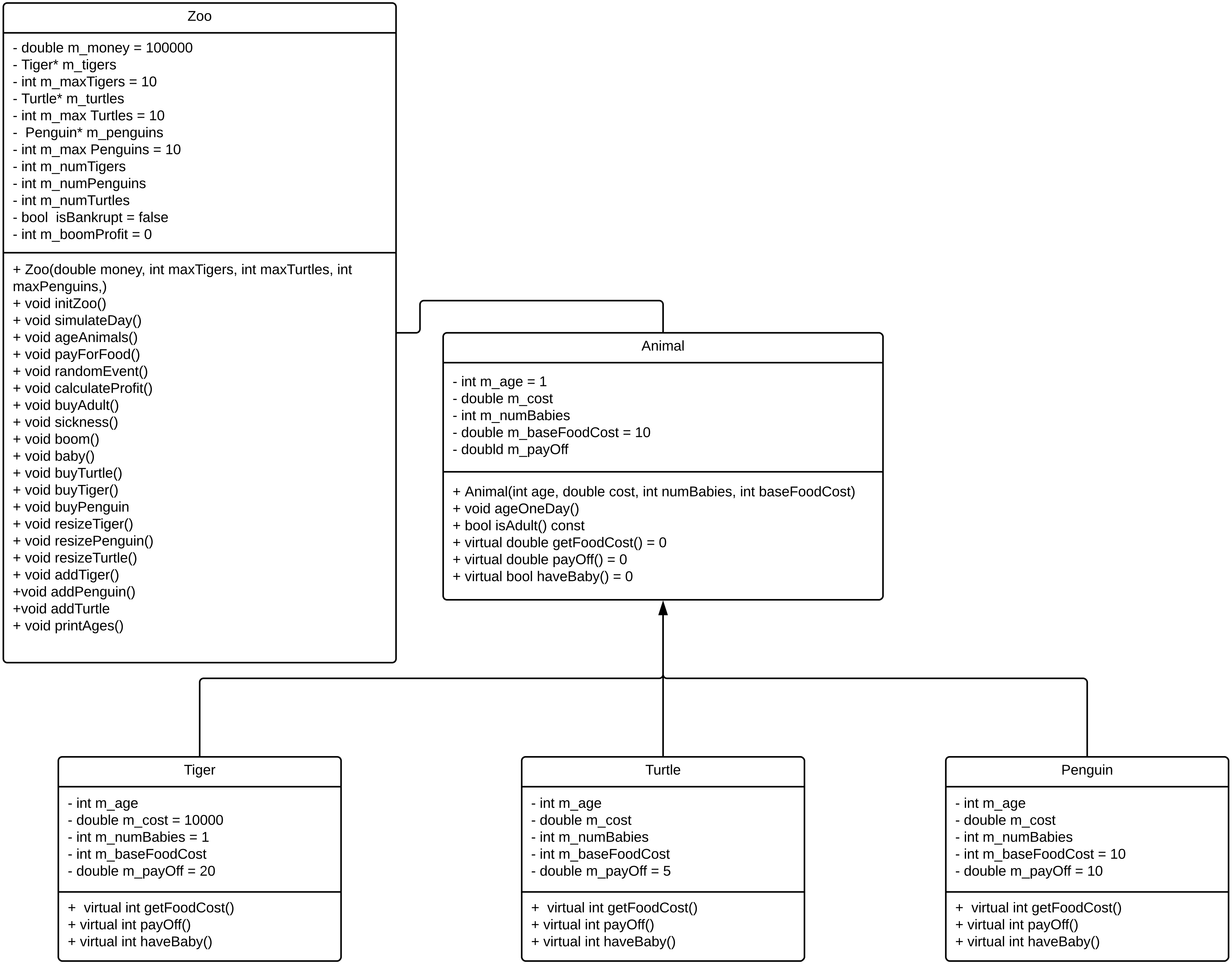
Project 2 Design and Reflection

Reflection

My original design was fairly effective this time, however, I did have to make several additions to it while trying to write the code. The first things I added that were not part of the original design were functions to add a new animal of each type, and functions to resize the animal arrays. My original design didn't account for how frequently this needed to be done (animals are added in the initZoo(), baby(), and buyAdult() functions). I decided to make dedicated functions for these operations to reduce redundant code.

I also made a significant change in how I implemented the action of having a baby. My initial design had haveBaby() member functions for each animal. I thought that this made sense, because the animal is the thing that is having the baby, however, I later realized that the constructor accomplishes everything that needs to happen when creating a baby when it is passed a starting age of 0. As such, I removed the Animal.haveBaby() function and implemented a Zoo member function for having babies that would add a new animal to the appropriate exhibit array with an age of 0 using Zoo::add(AnimalType) functions.

Another issue I had to address when trying to implement my original design had to do with default arguments for my constructor parameters. I decided to give each parameter in each of my constructors a default argument, however I ran into a situation where I wanted to pass a value for only two of my parameters, but the order that they were in didn't allow this. I chose the simple solution of reordering the parameters to fix this issue, however, if I had more time I would have implemented some overloaded constructors to address the different parameter combinations I might need if adding to this project.



Zoo

- double m_money = 100000
- Tiger* m_tigers
- int m_maxTigers = 10
- Turtle* m_turtles
- int m_max Turtles = 10
- Penguin* m_penguins
- int m_max Penguins = 10
- int m_numTigers
- int m_numPenguins
- int m_numTurtles
- bool isBankrupt = false
- int m_boomProfit = 0

+ Zoo(double money, int maxTigers, int maxTurtles, int maxPenguins,)
+ void initZoo()
+ void simulateDay()
+ void ageAnimals()
+ void payForFood()
+ void randomEvent()
+ void calculateProfit()
+ void buyAdult()
+ void sickness()
+ void boom()
+ void baby()
+ void buyTurtle()
+ void buyTiger()
+ void buyPenguin
+ void resizeTiger()
+ void resizePenguin()
+ void resizeTurtle()
+ void addTiger()
+void addPenguin()
+void addTurtle
+ void printAges()

Animal

- int m_age = 1
- double m_cost
- int m_numBabies
- double m_baseFoodCost = 10
- doubld m_payOff

+ Animal(int age, double cost, int numBabies, int baseFoodCost)
+ void ageOneDay()
+ bool isAdult() const
+ virtual double getFoodCost() = 0
+ virtual double payOff() = 0
+ virtual bool haveBaby() = 0

Tiger

- int m_age
- double m_cost = 10000
- int m_numBabies = 1
- int m_baseFoodCost
- double m_payOff = 20

+ virtual int getFoodCost()
+ virtual int payOff()
+ virtual int haveBaby()

Turtle

- int m_age
- double m_cost
- int m_numBabies
- int m_baseFoodCost
- double m_payOff = 5

+ virtual int getFoodCost()
+ virtual int payOff()
+ virtual int haveBaby()

Penguin

- int m_age
- double m_cost
- int m_numBabies
- int m_baseFoodCost = 10
- double m_payOff = 10

+ virtual int getFoodCost()
+ virtual int payOff()
+ virtual int haveBaby()

Design

```
Animal class functions:

Animal::Animal(int age = 1, double cost = 0, int numBabies = 0,
               int baseFoodCost = 10, double payOff = 0)
      The constructor will initialize member variables to passed values

virtual Animal::~Animal()

void Animal::ageOneDay()
      increment m_age

bool Animal::isAdult()
      return true if animal is 3 days or older
      return false otherwise

virtual int Animal::getFoodCost() = 0
      pure virtual function for getting cost of food

virtual int Animal::getPayOff() = 0
      pure virtual function for getting payoff amount

virtual int Animal::haveBaby() = 0

Tiger class functions:

Tiger::Tiger(int age = 1, double cost = 10000, int numBabies = 0,
   int baseFoodCost = 10, double payOff = .20) :
      Animal(age, cost, numBabies, baseFoodCost, payOff)

virtual Tiger::~Tiger()

virtual int Tiger::getFoodCost()
      return 5 times the base food cost

virtual int Tiger::getPayOff()
      return 20% of cost of animal
```

Penguin class functions:

```
Penguin::Penguin(int age = 1, double cost = 1000, int numBabies = 5,
   int baseFoodCost = 10, double payOff = .10) :
      Animal(age, cost, numBabies, baseFoodCost, payOff)

virtual Penguin::~Penguin()

virtual int Penguin::getFoodCost()
      return the base food cost

virtual int Penguin::getPayOff()
      return 10% of cost of animal
```

Turtle class functions:

```
Turtle::Turtle(int age = 1, double cost = 100, int numBabies = 10,
   int baseFoodCost = 10, double payOff = .05) :
      Animal(age, cost, numBabies, baseFoodCost, payOff)

virtual Turtle::~Turtle()

virtual int Turtle::getFoodCost()
      return the 50% base food cost

virtual int Turtle::getPayOff()
      return 5% of cost of animal
```

Zoo class functions:

```
Zoo::Zoo(int money = 100000, int maxTigers = 10, int maxPenguins = 10, int
maxTurtles = 10)
      initialize member variables to passed values.

      create three dynamic arrays for Tigers Penguins and Turtles, each with
      a number of elements equal to the maximum number of that type of
      animal.

      seed rand()

Zoo::~Zoo()
      Deallocate animal arrays

void Zoo::initZoo()
      const int minStartingAnimals = 1
      const int maxStartingAnimals = 2

      Ask user how animals of each animal type they want (1 or 2) add that
      many of each animal to the zoo and subtract the cost from the bank

void Zoo::simulateDay()
      do
            call ageAnimals() to increase animal ages by 1 day
            call payForFood() to subtract food cost from money
            call randomEvent() to simulate random even
            call calculateProfit() update money based on profit
            call buyAdult() ask user if they want to buy adult animal
      while(continue() and money > 0) ;
      if(money <= 0)
            print you lose!

void Zoo::ageAnimals()
      call ageOneDay() for each animal

void Zoo::payForFood()
      multiply cost of food for single animal by the number of animals of
      that type to get the cost of food for that animal.

      add the food cost totals for each animal and subtract it from money

      if(money <= 0)
            set isBankrupt = true
            set money = 0

void Zoo::randomEvent()
      get random number between 1 and 4

      switch (random number)
      case1: call sickness()
      case2: call boom()
      case3: call baby()
```

```
void Zoo:: sickness()
      generate random number 1 to 3
      switch (random number)
      case1: delete a Tiger
      case2: delete a Penguin
      case3: delete a Turtle

void Zoo::boom()
      boomFactor = random number 250-500
      m_boomProfit = boomFactor (number of Tigers)


void Zoo::baby()
      babyType = generate random number 1 to 3
      numChecked = 0
      hadBaby = false

      while(numChecked < 3 and !hadBaby)
            numChecked += 1;
            switch(random)
            case1:
                  if(Tiger.haveBaby())
                        hadBaby = true
                        create new tiger babys (age 0)
                        print A tiger had __ babies
                  else
                        random = random + 2 % 3
            case2:
                  if(Penguin.haveBaby())
                        hadBaby = true
                        create new penguin babys (age 0)
                        print A penguin had __ babies
                  else
                        random = random + 2 % 3
            case3:
                  if(Turtle.haveBaby())
                        hadBaby = true
                        create new turtle babys (age 0)
                        print A turtle had __ babies
                  else
                        random = random + 2 % 3
      if(!hadBaby)
            print No one was old enough to have a baby

Zoo::calculateProfit()
      Tiger profit = num tigers times profit per tiger
      Penguin profit = num penguins times profit per penguin
      Turtle profit = num turtles times profit per turtle

      totalProfit = tigerProfit + penguinProfit + turtleProfit + m_boomProfit

      m_boomProfit = 0
      m_money += totalProfit
```

```
Zoo::buyAdult()
      made purchase = false
      while madePurchase == false
            ask user to enter a number if they want to buy an adult
         0. is no
         1. is tiger
         2. is penguin
         3. is turtle

         switch(userInput)
            case0:
                  madePurchase = true
            case1:
                  if(money – costoftiger >= 0)
                        buyTiger()
            case2:
                  if(money – costofpenguin >= 0)
                        buyPenguin()
            case3:
                  if(money – costofturtle >=0)
                        buyTurtle()

Zoo::continue()
      ask user to enter 1 to continue and 2 to quit
```

Test Plan and Results

Element: Zoo element: Tiger

Test Case	**Input**	**Expected Output**	**Observed Output**
Test Zoo::Zoo() functionality and memory management.	Create Zoo object and let if go out of scope. Test with valgrind.	3 memory allocations and 3 frees no errors	3 memory allocations and 3 frees no errors
Test Zoo::addTiger()	Create Zoo object and call zoo.addTiger() twice. Cout the zoo object after each call.	The number of tigers should increase by one each time the function is called.	The number of tigers should increase by one each time the function is called.

Test Zoo::addTiger() edge case: tiger array is full	Create Zoo object and call zoo.addTiger() 11 times. Cout the zoo object after each call. The 11th call should cause the array to double in size and the number of tigers to increase by 1.	m_maxTigers = 20
		m_numTigers = 11
			m_maxTigers = 20
			m_numTigers = 11

Test Zoo::addPenguin()	Create Zoo object and call zoo.addPenguin() twice. Cout the zoo object after each call.	The number of Penguins should increase by one each time the function is called.	The number of Penguins should increase by one each time the function is called.

Test Zoo::addPenguin() edge case: Penguin array is full	Create Zoo object and call zoo.addPenguin() 11 times. Cout the zoo object after each call. The 11th call should cause the array to double in size and the number of Penguins to increase by 1.	m_maxPenguins = 20
		m_numPenguins = 11
			m_maxPenguins = 20
			m_numPenguins = 11

Test Zoo::addTurtle()	Create Zoo object and call zoo.addTurtle() twice. Cout the zoo object after each call.	The number of Turtles should increase by one each time the function is called.	The number of Turtles should increase by one each time the function is called.

Test Zoo::addTurtle() edge case: Turtle array is full	Create Zoo object and call zoo.addTurtle() 11 times. Cout the zoo object after each call. The 11th call should cause the array to double in size and the number of Turtles to increase by 1.	m_maxTurtles = 20
		m_numTurtles = 11
			m_maxTurtles = 20
			m_numTurtles = 11

Zoo::initZoo()	call zoo.initZoo() on newly created Zoo object
	Enter following when prompted:
	number of tigers: 2
	number of penguins: 1
	number of turtles:2
	zoo.m_money = 100000
		zoo.m_numTigers = 2
		zoo.m_numPenguins = 1
		zoo.m_numTurtles = 2
		zoo.m_money = 78800
			zoo.m_numTigers = 2
			zoo.m_numPenguins = 1
			zoo.m_numTurtles = 2
			zoo.m_money = 78800

Zoo::initZoo()	attempt to enter a starting numbers of tigers, penguins, and turtles that are not 1 or 2	the user is prompted to enter a number between 1 and 2 until they do	the user is prompted to enter a number between 1 and 2 until they do
Zoo::ageAnimals()	All ages are 1 call zoo.ageAnimals()	all animal ages are 2	all animal ages are 2
Zoo::payForFood()	Zoo object with two of each animal m_money = 77800 call zoo.payForFood()	m_money = 77670	m_money = 77670

Zoo::sickness()	call zoo.sickness() 10 times
	m_numTigers = 2
	m_numPenguins = 2
	m_numTurtles = 2
		each call removes an animal of a random type if there is one available	each call removes an animal of a random type if there is one available

Zoo::boom()	call zoo.boom() m_numTigers = 2	m_boomProfit = 2 * random number between 250 and 300	m_boomProfit = 2 * random number between 250 and 300
Zoo::baby()	call zoo.baby() with an adult animal in the zoo	a randomly selected adult animal has a baby	a randomly selected adult animal has a baby
Zoo::baby()	call zoo.baby() with no adult animals in the zoo	no baby is had	no baby is had
Zoo::randomEvent()	call zoo.randomEvent() 10 times	called sickness(), baby(), or boom() or printed that nothing happened	called sickness(), baby(), or boom() or printed that nothing happened
Zoo::calculateProfit()	zoo with 2 tigers 2 penguins and 2 call zoo.calculateProfit()	m_money increases by 4210	m_money increases by 4210

Zoo::simulateDay	call zoo.simulateDay() with print statements placed inside of ageAnimals() payForFood() randomEvent() calculateProfit() and buyAdult() to ensure they are executing	the following functions execute:
		ageAnimals()
		payForFood()
		randomEvent()
		calculateProfit()
		buyAdult()
			the following functions execute:
			ageAnimals()
			payForFood()
			randomEvent()
			calculateProfit()
			buyAdult()

Element: ZooTycoon element: Tiger

Test Case	Input	Expected Output	Observed Output
play game with normal inputs	choose 2 of each starting animal buy a tiger continue playing	Zoo is created with user specified number of each animal the zoo's money decreases by the cost of a tiger and the number of tigers goes up by one the game continues when the user chooses to continue	Zoo is created with user specified number of each animal the zoo's money decreases by the cost of a tiger and the number of tigers goes up by one the game continues when the user chooses to continue
run out of money	continue buying tigers until money <=0	game ends and prints that the user ran out of money and lost	game ends and prints that the user ran out of money and lost
quit	choose to quit when prompted	game ends	game ends

Element: Animal element: Tiger

Test Case	Input	Expected Output	Observed Output
Crete Animal object	Animal(1,10)	Animal object with age of 1 and base food cost of 10	Animal object with age of 1 and base food cost of 10

Element: Tiger element: Tiger

Test Case	Input	Expected Output	Observed Output
Crete Tiger object	Tiger(1,10)	Tiger object with age of 1 and base food cost of 10	Tiger object with age of 1 and base food cost of 10

Element: Penguin element: Tiger

Test Case	Input	Expected Output	Observed Output
Crete Penguin object	Penguin(1,10)	Penguin object with age of 1 and base food cost of 10	Penguin object with age of 1 and base food cost of 10

Element: Turtle element: Tiger

Test Case	Input	Expected Output	Observed Output
Crete Turtle object	Turtle(1,10)	Turtle object with age of 1 and base food cost of 10	Turtle object with age of 1 and base food cost of 10

Element: Input Validation

Test Case	Input	Expected Output	Observed Output
input within range. Range is positive to positive	int test = intInputValidation(7, 10) enter 6	test = 6	test = 6
input above range. Range is positive to positive	int test = intInputValidation(7, 10) enter 11	requests that user enters a different number	requests that user enters a different number
input below range. Range is positive to positive	int test = intInputValidation(7, 10) enter 5	requests that user enters a different number	requests that user enters a different number
input within range. Range is negative to positive	int test = intInputValidation(-7, 10) enter 6	test = 6	test = 6
input above range. Range is negative to positive	int test = intInputValidation(-7, 10) enter 11	requests that user enters a different number	requests that user enters a different number
input below range. Range is negative to positive	int test = intInputValidation(-7, 10) enter -8	requests that user enters a different number	requests that user enters a different number
input within range. Range is negative to negative	int test = intInputValidation(-7, -1) enter -5	test = -5	test = -5
input above range. Range is negative to negative	int test = intInputValidation(-7, -1) enter 11	requests that user enters a different number	requests that user enters a different number
input below range. Range is negative to negative	int test = intInputValidation(-7, -1) enter -8	requests that user enters a different number	requests that user enters a different number