# Name: Hamad Nasir

# Roll Number: 120312

# Section: BSCS-6C

# Lab-4

## Task 2

## Code

\*Task 2\*

package savingsaccount;

public class SavingsAccount { // main class

static double annualInterestRate = 0.00; // can be called outside class without creating object

private double savingsBalance; //can only be called inside class

public SavingsAccount() { // constructuctors

savingsBalance = 1000.00;

annualInterestRate = 0.00;

}

public SavingsAccount(double initBalance) {

if(initBalance >= 1000.00)

savingsBalance = initBalance;

else {

System.out.println("The initial balance is below $1000.00");

savingsBalance = 0.00;

}

}

static void modifyInterestRate(double interestRate) { //mettod with no return type

annualInterestRate = interestRate;

}

public void calculateMonthlyInterest() { //method calculates

savingsBalance = savingsBalance + (savingsBalance \* annualInterestRate / 12);

}

public double getBalance() {

return savingsBalance;

}

public static void main(String args[]) {

SavingsAccount saver1 = new SavingsAccount(2000);

SavingsAccount saver2 = new SavingsAccount(3000);

SavingsAccount.modifyInterestRate(.04);

// prints the initial balance

System.out.println("The initial balance for saver 1 is: " + saver1.getBalance());

System.out.println("The initial balance for saver 2 is: " + saver2.getBalance());

System.out.println();

saver1.calculateMonthlyInterest();

saver2.calculateMonthlyInterest();

// prints the amount at interest rate of 4%

System.out.printf("\nSaver1 balance at interest of 4%% is: %.2f: " , saver1.getBalance());

System.out.printf("\nSaver2 balance at interest of 4%% is: %.2f: " , saver2.getBalance());

System.out.println();

SavingsAccount.modifyInterestRate(.05);

saver1.calculateMonthlyInterest();

saver2.calculateMonthlyInterest();

// prints the amount at interest rate of 5%

System.out.printf("\nSaver1 balance at interest of 5%% is: %.2f: " , saver1.getBalance());

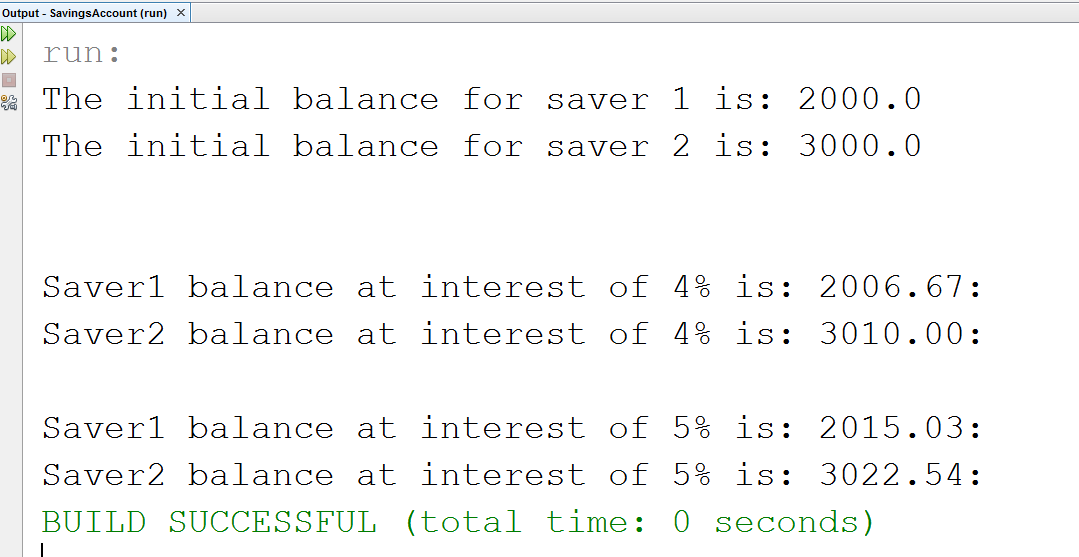
System.out.printf("\nSaver2 balance at interest of 5%% is: %.2f: " , saver2.getBalance());

System.out.println();

}

}

# Output:



# Task 1

# Code

\*Task 1\*

package perfectnumber;

public class PerfectNumber {

public static void main(String[] args) {

// loop and print perfect numbers between 1 and 1000 using our perfectNum() method

for (int number = 1; number <= 1000; number++) {

if (isPerfect(number)) {

System.out.printf("\nThe number %d is a perfect number! \n Factors:",number);

System.out.println();

for(int a=1;a<number;a++){

if(number%a==0)

System.out.printf(" %d",a);

}

}

}

} // end main()

// Method

public static Boolean isPerfect(int number) {

// declare a variable to hold the sum of number's factors

int sum = 0;

// loop over all the numbers up to 'number' to determine if they are factors

// if they are then add them to 'sum'

for (int factor = 1; factor < number; factor++) {

if (number % factor == 0) {

sum += factor;

}

}

// if 'sum' is equal to the number passed in then it's a perfect number

if (sum == number)

{

return true;

} else {

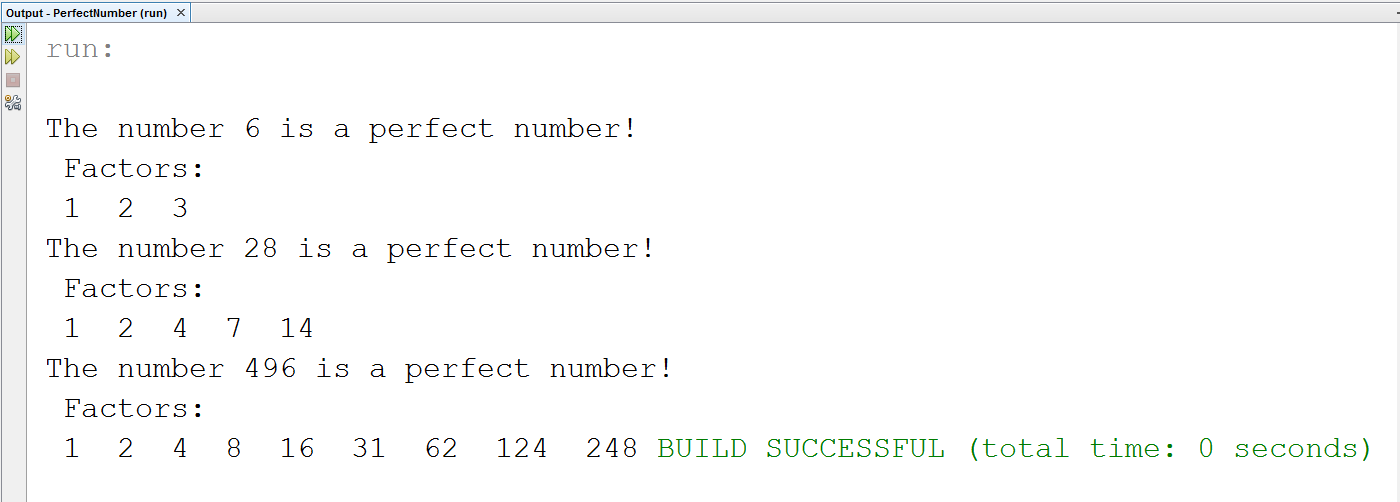
return false;

}

} // end public method isPerfect()

} // end PerfectNumbers class

# Output:



# Task 3

# Code

## Class test

public class Test {

public static void main(String[] args){

Time2 clock = new Time2(0,0,0);

for(int second = 0; second <= 86400; second++){

clock.tick();

System.out.println(clock);

}

}

}

## Class Time2

public class Time2

{

private int hour; // 0 - 2

private int minute; // 0 - 59

private int second; // 0 - 59

// Time2 no-argument constructor:

// initializes each instance variable to zero

public Time2()

{

this( 0, 0, 0 ); // invoke Time2 constructor with three arguments

} // end Time2 no-argument constructor

// Time2 constructor: hour supplied, minute and second defaulted to 0

public Time2( int h )

{

this( h, 0, 0 ); // invoke Time2 constructor with three arguments

} // end Time2 one-argument constructor

// Time2 constructor: hour and minute supplied, second defaulted to 0

public Time2( int h, int m )

{

this( h, m, 0 ); // invoke Time2 constructor with three arguments

} // end Time2 two-argument constructor

// Time2 constructor: hour, minute and second supplied

public Time2( int h, int m, int s )

{

setTime( h, m, s ); // invoke setTime to validate time

} // end Time2 three-argument constructor

// Time2 constructor: another Time2 object supplied

public Time2( Time2 time )

{

// invoke Time2 three-argument constructor

this( time.getHour(), time.getMinute(), time.getSecond() );

} // end Time2 constructor with a Time2 object argument

// Set Method

// set a new time value using universal time;

// validate the data

public void setTime( int h, int m, int s )

{

setHour( h ); // set the hour

setMinute( m ); // set the minute

setSecond( s ); // set the second

} // end method setTime

// validate and set hour

public void setHour( int h )

{

if ( h >= 0 && h < 24 )

hour = h;

else

throw new IllegalArgumentException( "hour must be 0-23" );

} // end method setHour

// validate and set minute

public void setMinute( int m )

{

if ( m >= 0 && m < 60 )

minute = m;

else

throw new IllegalArgumentException( "minute must be 0-59" );

} // end method setMinute

// validate and set second

public void setSecond( int s )

{

if ( s >= 0 && s < 60 )

second = ( ( s >= 0 && s < 60 ) ? s : 0 );

else

throw new IllegalArgumentException( "second must be 0-59" );

} // end method setSecond

// Get Methods

public int getHour(){

return hour;

}

public int getMinute(){

return minute;

}

public int getSecond(){

return second;

}

// convert to String in universal-time format (HH:MM:SS)

public String toUniversalString()

{

return String.format("%02d:%02d:%02d", getHour(), getMinute(), getSecond() );

}

// convert to String in standard-time format (H:MM:SS AM or PM)

public String toString()

{

return String.format( "%d:%02d:%02d %s",

( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),

getMinute(), getSecond(), ( getHour() < 12 ? "AM" : "PM" ) );

} // end method toString

public void incrementMinute(){

minute++;

if(minute == 60){

incrementHour(); //incrementing to the next hour if minutes become 60

minute = 0; //reseting the minutes

}

}

public void incrementHour(){

hour++;

if(hour == 24){

hour = 0; //reseting the hours

}

}

public void tick(){

second++;

if(second == 60){

incrementMinute(); //incrmenting by one hour

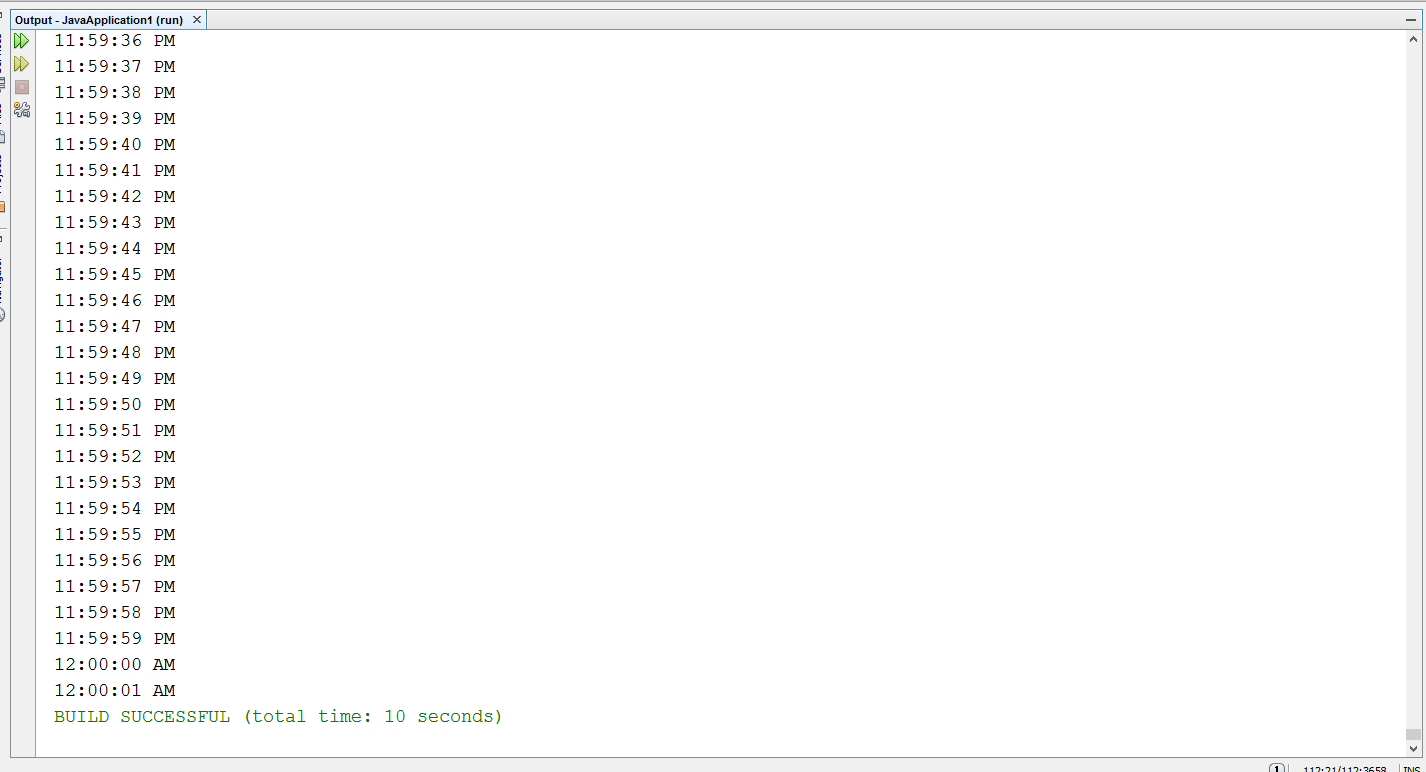
second = 0; //reseting the seconds

}

}

} // end class Time2

# Output:



# ACTIVITY #1

# Code

# Wrong

class Time

{

private int hour; // 0 - 23

private int minute; // 0 - 59

private int second; // 0 - 59

// TODO: Add constructors definitions here

// set a new time value using universal time; throw an

// exception if the hour, minute or second is invalid

// TODO: Add constructors definitions here

Time ()//NO ARGS CONSTRUCTOR

{

setTime( 0, 0, 0 );

}

Time (int h)//ONLY HOUR CONSTRUCTOR

{

setTime( h, 0,0 );

}

Time (int h,int m)//HOUR AND MINUTE CONSTRUCTOR

{

setTime( h, m, 0);

}

public void setTime( int h, int m, int s )

{

// validate hour, minute and second

if ( ( h >= 0 && h < 24 ) && ( m >= 0 && m < 60 ) && ( s >= 0 && s < 60 ) )

{

hour = h;

minute = m;

second = s;

} // end if

else

throw new IllegalArgumentException("hour, minute and/or second was out of range" );

} // end method setTime

// convert to String in universal-time format (HH:MM:SS)

public String toUniversalString()

{

return String.format( "%02d:%02d:%02d", hour, minute, second );

} // end method toUniversalString

// convert to String in standard-time format (H:MM:SS AM or PM)

public String toString()

{

return String.format( "%d:%02d:%02d %s",

( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),

minute, second, ( hour < 12 ? "AM" : "PM" ) );

} // end method toString

} // end class Time

// This class tests the Time Class by creating different objects.

public class TimeTest{

public static void main(String [] args){

Time t0 = new Time(); // Set Time to 00:00:00

Time t1 = new Time(11); // Set Time to 11:00:00

Time t2 = new Time(12, 40); // Set Time to 12:40:00

Time t3 = new Time(23, 40, 55); // Set Time to 23:40:55

Time t4 = new Time(23, 40, 65); // Set Time to 23:40:65

// Print All Times in Universal Format

System.out.println(t0.toUniversalString());

System.out.println(t1.toUniversalString());

System.out.println(t2.toUniversalString());

System.out.println(t3.toUniversalString());

// Print All Times in Standard Format

System.out.println(t0);

System.out.println(t1);

System.out.println(t2);

System.out.println(t3);

}

}

# OUTPUT: With wrong arguments pass

# Corrected Code

class Time

{

private int hour; // 0 - 23

private int minute; // 0 - 59

private int second; // 0 - 59

// TODO: Add constructors definitions here

Time ()//NO ARGS CONSTRUCTOR

{

setTime( 0, 0, 0 );

}

Time (int h)//ONLY HOUR CONSTRUCTOR

{

setTime( h, 0,0 );

}

Time (int h,int m)//HOUR AND MINUTE CONSTRUCTOR

{

setTime( h, m, 0);

}

Time(int h,int m,int s)//THREE ARGUMENTS CONSTRUCTOR

{

setTime( h, m, s );

}

// set a new time value using universal time; throw an

// exception if the hour, minute or second is invalid

public void setTime( int h, int m, int s )

{

// validate hour, minute and second

if ( ( h >= 0 && h < 24 ) && ( m >= 0 && m < 60 ) && ( s >= 0 && s < 60 ) )

{

hour = h;

minute = m;

second = s;

} // end if

else

throw new IllegalArgumentException("hour, minute and/or second was out of range" );

} // end method setTime

// convert to String in universal-time format (HH:MM:SS)

public String toUniversalString()

{

return String.format( "%02d:%02d:%02d", hour, minute, second );

} // end method toUniversalString

// convert to String in standard-time format (H:MM:SS AM or PM)

public String toString()

{

return String.format( "%d:%02d:%02d %s",

( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 ),

minute, second, ( hour < 12 ? "AM" : "PM" ) );

} // end method toString

} // end class Time// end class Time

// This class tests the Time Class by creating different objects.

public class TimeTest{

public static void main(String [] args){

Time t0 = new Time(); // Set Time to 00:00:00

Time t1 = new Time(11); // Set Time to 11:00:00

Time t2 = new Time(12, 40); // Set Time to 12:40:00

Time t3 = new Time(23, 40, 55); // Set Time to 23:40:55

Time t4 = new Time(23, 40, 33); // Set Time to 23:40:33

// Print All Times in Universal Format

System.out.println(t0.toUniversalString());

System.out.println(t1.toUniversalString());

System.out.println(t2.toUniversalString());

System.out.println(t3.toUniversalString());

// Print All Times in Standard Format

System.out.println(t0);

System.out.println(t1);

System.out.println(t2);

System.out.println(t3);

}

}

# Output: With correct arguments pass

# ACTIVITY #2

// Demonstrate method overloading.

class Overload {

void test() {

System.out.println("No parameters");

}

// Overload test for one integer parameter.

void test(int a) {

System.out.println("a: " + a);

}

// Overload test for two integer parameters.

void test(int a, int b) {

System.out.println("a and b: " + a + " " + b);

}

// Overload test for a double parameter

double test(double a) {

System.out.println("double a: " + a);

return a\*a;

}

}

public class OverloadTest {

public static void main(String args[]) {

Overload ol = new Overload();

double result;

// call all versions of test()

ol.test();

ol.test(10);

ol.test(10, 20);

result = ol.test(123.2);

System.out.println("Result of ol.test(123.2): " + result);

}

}

# OUTPUT:

# ACTIVITY #3

# Code

void test(int a) {

System.out.println("a: " + a);

}

// Overload test for one integer parameter.

int test(int a) {

System.out.println("a: " + a);

return 0;

}

**The overloading of parametric function is not allowed because same parameter function repetition is not allowed, although they have different return type but this cannot affect.**

void test(int a, int b) {

System.out.println("a and b: " + a + " " + b);

}

// Overload test for two integer parameters.

void test(int b, int a) {

System.out.println("a and b: " + a + " " + b);

**Here the repetition of number of parameters takes place so it is not allowed we have to change the type of parameters in order to correct it.**