# How to calculate Birthday

//New feature LocalDate in jdk 1.8

**import** java.time.LocalDate;

**import** java.time.Period;

//year, month and date are instance variable

**public** **int** calculateAge() {

**if** (year == 0 || month == 0 || date == 0) {

**return** 0;

}

**else** {

// calculate birthday

LocalDate today = LocalDate.*now*();

LocalDate birthday = LocalDate.*of*(year, month, date);

Period p = Period.*between*(birthday, today);

**return** p.getYears();

}

}

# How to input Date

Date myDate = **new** Date(1979, 12, 31);

System.***out***.println(myDate.getYear());

System.***out***.println(myDate.getMonth());

System.***out***.println(myDate.getDate());

# How to convert int to double

**double** heightInMeters = (**double**) height/100;

**return** weight/(heightInMeters\*heightInMeters);

# How to round

Int I = (int) Math.round(double i);

# Use other package

//Current package name

**package** Exercise\_3\_17;

//Other package name+Class name

**import** Exercise\_3\_16.HeartRates;

# How to get individual number

Scanner s = **new** Scanner(System.***in***);

System.***out***.println("Input thousand value:");

**int** nbr = s.nextInt();

**int** [] x={0,0,0,0};

**int** n=3;

**while** (nbr>0){

x[n]=nbr%10;

nbr/=10;

System.***out***.println(x[n]);

n--;}

# Reset value when use Nested while or for

**int** a = 1, b = 1, c = 1;

**while** (a <= 500) {

**while** (b <= 500) {

**while** (c <= 500) {

**if** (a \* a == b \* b + c \* c) {

System.***out***.print("This is Pythagoran Triples a = " + a

+ " b = " + b + " c = " + c + "\n");

}

c++;

}

c = 1;

b++;

}

b = 1;

a++;

}

# Calculate 3rd side of triangle

Math.*hypot*(a, b));

# Manually calculate exponent

**private** **static** **int** calculateIntergerPower(**int** base, **int** exponent){

**int** value = 1;

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

value\*=base;

}

**return** value;

# Get integer length

int length = String.valueOf(1000).length();

# GCD

In [mathematics](http://en.wikipedia.org/wiki/Mathematics), the **greatest common divisor** (**gcd**), also known as the **greatest common factor** (**gcf**), **highest common factor** (**hcf**), or **greatest common measure** (**gcm**), of two or more [integers](http://en.wikipedia.org/wiki/Integer) (when at least one of them is not zero), is the largest positive integer that [divides](http://en.wikipedia.org/wiki/Divisor) the numbers without a [remainder](http://en.wikipedia.org/wiki/Remainder). For example, the GCD of 8 and 12 is 4.

Use Euclid’s algorithm to calculate GCD

# Convert String to Int, convert Int to String, define String format

Integer.parseInt(String value)

Double.parseDouble(String value)

Integer value.toString()

String.format(“%d:%”, mystring)

public String toString()

{

return String.format( "%s, %s Hired: %s Birthday: %s",lastName, firstName, hireDate, birthDate );

}

# Use Args, multi dimentions and convert String to double

**public** **class** EnhancedFor {

/\*\*

\* **@param** args

\*/

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

// **TODO** Auto-generated method stub

**if** (args.length == 0) {

System.***out***.println("Enter Double values");

} **else** {

System.***out***.println("The product is :" + *calculateProduct*(args));

}

}

**static** **double** calculateProduct(String... args) {

**double** product = 1.0;

**for** (String value : args) {

product \*= Double.*parseDouble*(value);

}

**return** product;

}

}

# This with instance variables

public class Time2{

private int hour; // 0 - 23

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

# Date, Month and Year

**package** Exercise\_8\_8;

//Fig. 8.7: Date.java

//Date class declaration.

**public** **class** Date {

**private** **int** month; // 1-12

**private** **int** day; // 1-31 based on month

**private** **int** year; // any year

// constructor: call checkMonth to confirm proper value for month;

// call checkDay to confirm proper value for day

**public** Date(**int** theMonth, **int** theDay, **int** theYear) {

month = checkMonth(theMonth); // validate month

year = theYear; // could validate year

day = checkDay(theDay); // validate day

System.***out***.printf("Date object constructor for date %s\n", **this**);

} // end Date constructor

**public** **void** NextDay(){

**int** newDay = day+1;

**if** (checkDay(newDay)!=1){

day++;

}

**else** {

day=1;

**int** newMonth = month +1;

**if** (checkMonth(newMonth)!=1){

month++;

}

**else** {

month=1;

year++;

}

}

}

// utility method to confirm proper month value

**private** **int** checkMonth(**int** testMonth) {

**if** (testMonth > 0 && testMonth <= 12) // validate month

**return** testMonth;

**else** // month is invalid

{

System.***out***.printf("Invalid month (%d) set to 1.", testMonth);

**return** 1; // maintain object in consistent state

} // end else

} // end method checkMonth

// utility method to confirm proper day value based on month and year

**private** **int** checkDay(**int** testDay) {

**int** daysPerMonth[] = { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,

31 };

// check if day in range for month

**if** (testDay > 0 && testDay <= daysPerMonth[month])

**return** testDay;

// check for leap year

**if** (month == 2 && testDay == 29

&& (year % 400 == 0 || (year % 4 == 0 && year % 100 != 0)))

**return** testDay;

System.***out***.printf("Invalid day (%d) set to 1.", testDay);

**return** 1; // maintain object in consistent state

} // end method checkDay

// return a String of the form month/day/year

**public** String toString() {

**return** String.*format*("%d/%d/%d", month, day, year);

} // end method toString

} // end class Date

# Throw Exception in If statement

if ( salary >= 0.0 )

baseSalary = salary;

else

throw new IllegalArgumentException("Base salary must be >= 0.0" );

# Retrieve Class Name

for ( int j = 0; j < employees.length; j++ )

System.out.printf("Employee %d is a %s\n",j,employees[ j ].getClass().getName() );

# Interface objects belong to subclass

**1** // Fig. 10.15: PayableInterfaceTest.java

**2** // Tests interface Payable.

**34**

public class PayableInterfaceTest

**5** {

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

**7** {

**8** // create four-element Payable array

**9**

**10**

**11** // populate array with objects that implement Payable

**12** payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00 );

**13** payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95 );

**14** payableObjects[ 2 ] =

**15** new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00 );

**16** payableObjects[ 3 ] =

**17** new SalariedEmployee( "Lisa", "Barnes", "888-88-8888", 1200.00 );

**18**

**19** System.out.println(

**20** "Invoices and Employees processed polymorphically:\n" );

**21**

**22** // generically process each element in array payableObjects

**23** for ( Payable currentPayable : payableObjects )

**24** {

**25** // output currentPayable and its appropriate payment amount

**26** System.out.printf( "%s \n%s: $%,.2f\n\n",currentPayable.toString(),"payment due", currentPayable.getPaymentAmount());

**29** } // end for

**30** } // end main

**31** } // end class PayableInterfaceTest