Introduction to CS102

CS102 – Algorithms and Programming II

- Objectives
  - Undertake real-world design task
  - Work as a member of a team
  - Practice communication in written & oral form
  - Learn more programming techniques
  - Practice independent learning!
- General
  - Transform basic computer literacy, design and programming skills you learnt in CS101 into practice
  - Expand the range of techniques you have to solve problems

This course should help you…

- improve your programming abilities
  - Enhanced OOP
  - GUI & Event-driven programming
  - Recursion
  - Data structures

- practice core engineering skills
  - Written & oral communication
  - Teamwork
  - Independent learning

Components of the course

- Topics to be covered
  - Recursion
  - Files
  - Some basic data structures
  - Object-oriented programming
  - Event-driven architectures
  - Searching and sorting
- Project
  - Commercial-quality program
  - Fully documented
  - Bug-free and easy to use
  - Group project
  - Written reports and presentations (requirements, specifications, detailed design, user manuals)

Textbook

- Big Java 5th Edition International Student Version, Cay S. Horstmann, Wiley, 2014 (as for CS101)

Course - Organisation

- CS102 taken by all CS & EE students
  - (~350+) in 7 sections (4 instructors, lots of assistants!)
- 4 credits – 3hr lecture & 4hr lab every week
  - attend labs every week even though there may not always be assignments
- Two tracks…
  - Lectures & Labs (as per CS101)
  - Design project
    - Group & project selection
    - Requirements, UI design, Detailed Design
    - Implementation & Demo

Course Rules

- Grading (Tentative):
  - 15% - Lab assignments (& contributions to course forum)
  - 30% - Midterm Exam
  - 20% - Final Exam (see note below regarding examinations)
  - 15% - * Reports, Presentations & Participation
    {Requirements & User-Interface 10%, Detailed Design 5%}
  - 10% - * Demonstration, Final code & documentation
    { inc. wiki & peer grade! }
  - 10% - Homework & Quizzes

- Reasonable minimum grade required to pass course!

FZ Grade and Passing

- In line with the new university regulations, students who fail to meet the following minimum course work requirements will receive an FZ grade and not be eligible to enter the final exam.
- Minimum course requirements:
  - more than 35% on the midterm exam
  - more than 75% lab average
  - personal project logs properly completed each week
  - reasonable contributions to each stage of the project.

  - Makeup exam may be taken by students who miss the midterm exam for documented medical reasons. The makeup will be given before the deadline for notification of FZ grades. Makeups are not given for labs, projects, quizzes, homeworks, or the final exam.

Grading Scales

Labs

(100) Fully complete, correct and understood

(80) Almost fully complete, correct and/or understood

(20) Incomplete/incorrect, poor understanding, little real interest/effort shown

(0) no real attempt!

Projects

(10) excellent (almost impossible!)

(8) good

(6) ok but could be better

(4) weak definitely not up to scratch, more effort needed.

(0) no real attempt!

Students must upload whatever done 24 hours before lab. Get feedback, correct & resubmit during lab without penalty. Demonstrate understanding. The objective is to learn.

Course – Misc

- Lab sessions start on Week 3
- Moodle – check frequently!
  - Schedule
- See also (your section's webpage)
  - http://www.cs.bilkent.edu.tr/~erman/CS102.html
- Cheating/Plagiarism!

TODO

- Enroll to Moodle
- Lab assignment 1 (due in lab week 3)
- Find group & project (asap)
  - Same section only
  - 5 people

- Any questions?

Last Year’s Projects

Actiwiki

- This project is an android application which takes user's location and shows activities around there like theatres, concerts and sport competitions. Besides, it provides details for activities.

Simulate the Circuit

- This program is intended to provide a fun, elementary interface about electrical circuits for junior – high school kids who are new at exploring this area of physics. The design options will include adding basic circuit elements such as batteries and light bulbs and it will enable user to obtain information about properties of the customized circuit like voltage, current, and equivalent resistance. Designing circuits for different purposes and presenting the properties of the circuit will be easier and more educative for learners

Online Experiment for Aiding Psychological Studies

- This online experiment will consist of two parts, which are test and game parts. By writing test, we mean that there will be a quick common knowledge quiz from many categories. Questions will have a point reward in return if the user answers correctly. The application will record the answers and the results of the participants, these results will be considered in the upcoming sections of the application. Coming to the latter stage, participants will face an interface where they can negotiate for an imaginary wage, this part will evaluate the behavior of the contestants. Gathering information from these two parts, our application will lastly gather the demographic information from the user such as age, department, location and so on.

- All of this information will then be provided to the main user of the application to be considered in the area of Behavioral Economics.

Light Flow Game

- Light Flow is a single-player game which there exist optical instruments and light sources on a 2D game board, being created with the aim of providing students and teachers with both an enjoyable and an educational tool to learn about a physics topic, optics. The aim of the players is to direct the colored light beams to the matching-colored targets. To play the game, the players need to drag, drop and rotate the equipment which consists of lenses, mirrors and color filters implemented in the game. This game has unique features, which distinguish our game from the other optic games.
- As our game is created with the purpose of providing an educational tool, the game includes tutorial pages giving information about the optical instruments such as lenses, mirrors and color filters. The users are also provided with a level editor which enables them to create their own levels and to observe the interactions between the light beams and the optical instruments by placing light sources, mirrors, lenses and filters.

What has been covered in CS101

- Introduction
  - Computer processing, hardware components, etc.
- Syntax and Semantics of Java
  - Identifiers, assignments, precedence rules, …
  - Console input/output
  - String class
- Flow of control
  - Conditional statements
    - boolean expressions
    - if/else
    - switch
  - Loops
    - for
    - while
    - do while

What has been covered in CS101

- Classes and Objects
  - Instance variables and methods
  - Public/private variables
  - Constructors, mutators, accessors, copy constructors
  - Static methods and variables
  - Memory and references (deep/shallow copy)
- Arrays
  - Creating, accessing
  - Length
  - Bounds of array
  - Multidimensional arrays
  - Ragged arrays
- Enumerated types
- ArrayList/Generics

Syllabus

- Writing classes
  - Objects
  - UML Diagrams
  - Encapsulation
  - Determining the classes and objects that are needed for a program
  - Relationships that can exist among classes
  - Interfaces
  - Method design and method overloading
  - Inheritance, multiple inheritance
  - Class hierarchies
  - Overloading vs. overriding
  - Protected/super
  - Polymorphism

Syllabus

- Sorting
  - Selection
  - Insertion
  - Other sort algorithms
- Search
  - Linear
  - Binary
- Exceptions
  - Exception handling
  - Exception propagation
  - finally/throw statements
- Recursion
  - Recursive thinking
  - Recursive programming
  - Indirect recursion
- Introduction to abstract data types
  - Queues
  - Stacks
  - Trees

Syllabus

- GUI
  - How to use graphical user interface components
  - GUI layouts
  - Polygons and polylines
  - Events: mouse, keyboard…
  - Event adapter classes
  - Timer class
  - Sliders
  - Split panes
  - Scroll panes
  - Combo boxes
  - Recursion in graphics

Program Development

- The mechanics of developing a program include several activities
  - writing the program in a specific programming language (such as Java)
  - translating the program into a form that the computer can execute
  - investigating and fixing various types of errors that can occur
- Software tools can be used to help with all parts of this process

Language Levels

- There are four programming language levels:
  - machine language
  - assembly language
  - high-level language
- Each type of CPU has its own specific machine language
- The other levels were created to make it easier for a human being to read and write programs

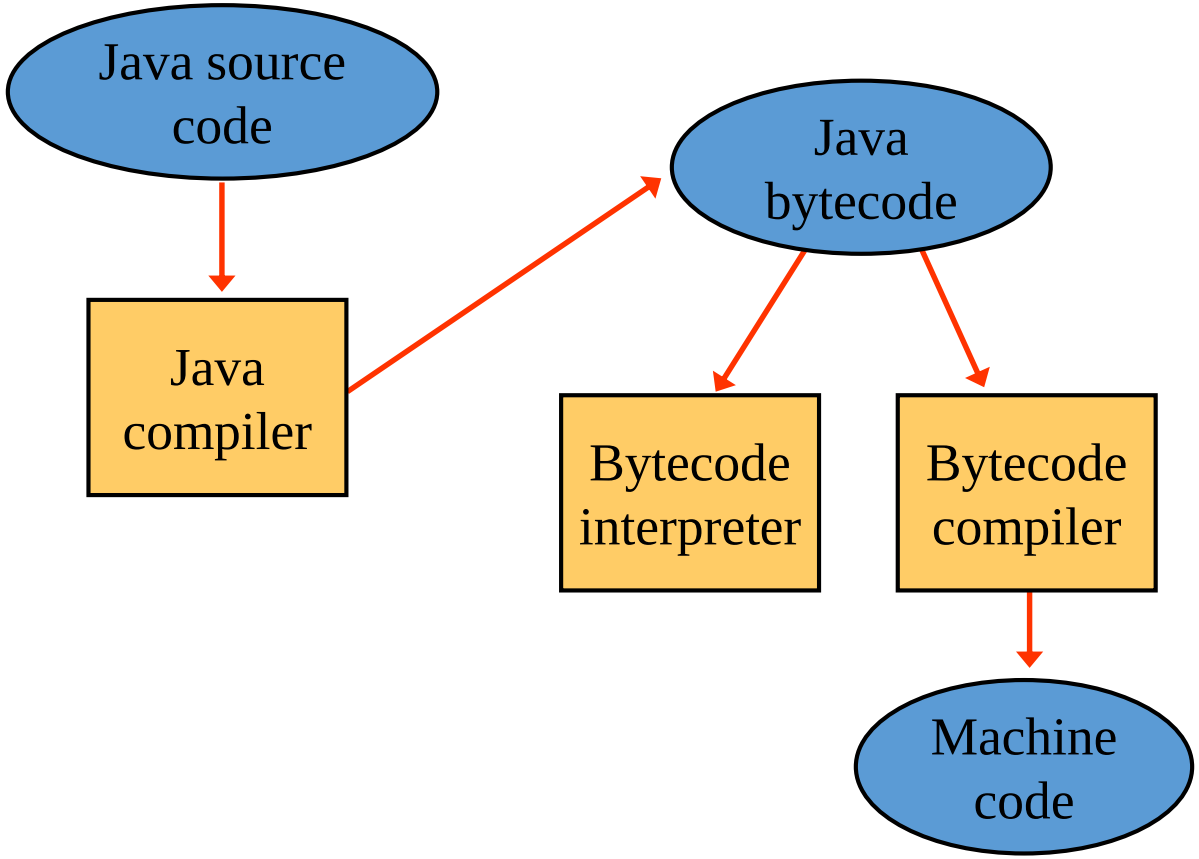
Programming Languages

- Each type of CPU executes only a particular machine language
- A program must be translated into machine language before it can be executed
- A compiler is a software tool which translates source code into a specific target language
- Often, that target language is the machine language for a particular CPU type
- The Java approach is somewhat different

Java Translation

- The Java compiler translates Java source code into a special representation called bytecode
- Java bytecode is not the machine language for any traditional CPU
- Another software tool, called an interpreter, translates bytecode into machine language and executes it
- Therefore the Java compiler is not tied to any particular machine
- Java is considered to be architecture-neutral

Java Translation



Development Environments

- There are many programs that support the development of Java software, including:
  - Sun Java Development Kit (JDK)
  - Sun NetBeans
  - IBM Eclipse
  - Borland JBuilder
  - MetroWerks CodeWarrior
  - BlueJ
  - jGRASP
- Though the details of these environments differ, the basic compilation and execution process is essentially the same

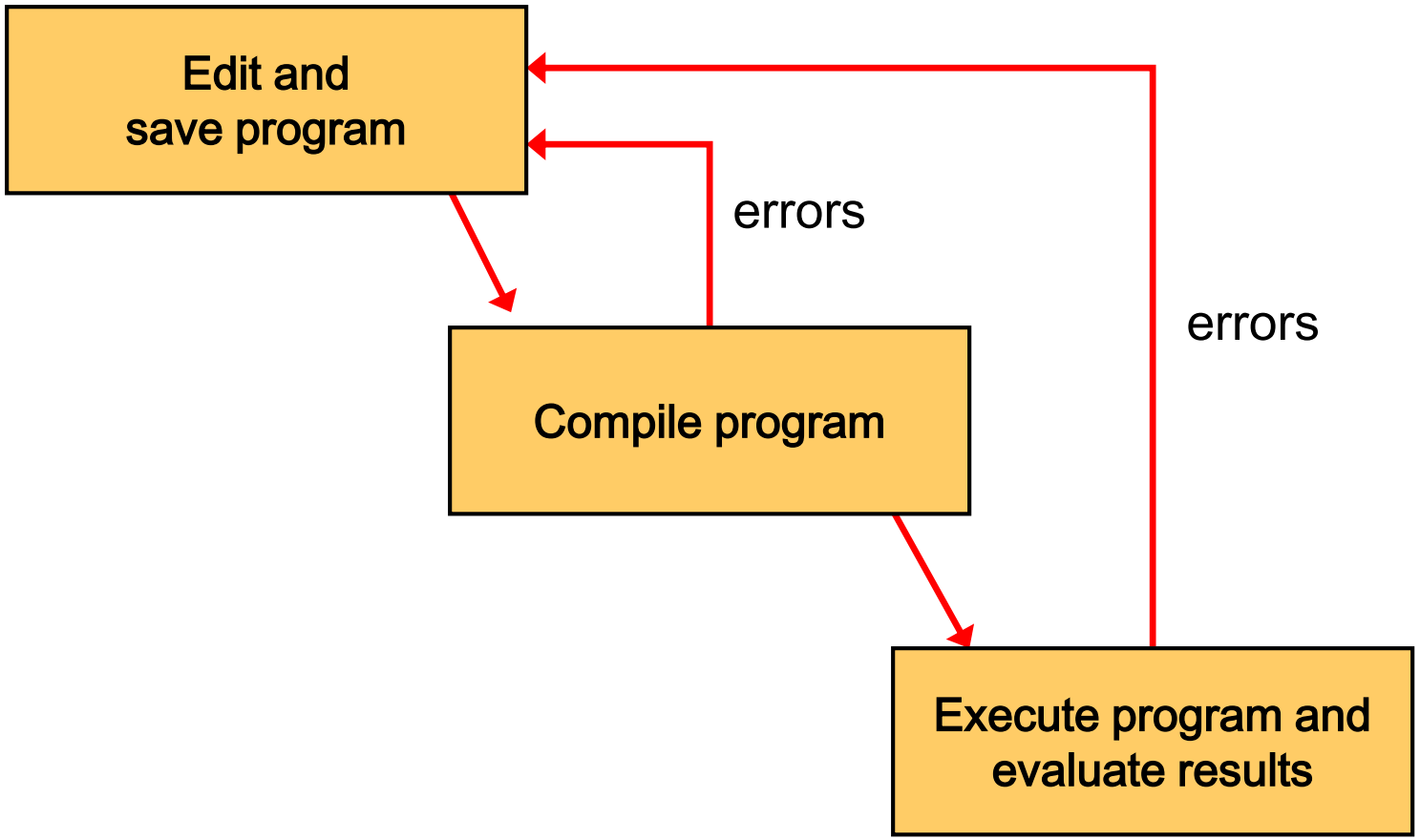
Syntax and Semantics

- The syntax rules of a language define how we can put together symbols, reserved words, and identifiers to make a valid program
- The semantics of a program statement define what that statement means (its purpose or role in a program)
- A program that is syntactically correct is not necessarily logically (semantically) correct
- A program will always do what we tell it to do, not what we meant to tell it to do

Errors

- A program can have three types of errors
- The compiler will find syntax errors and other basic problems (compile-time errors)
  - If compile-time errors exist, an executable version of the program is not created
- A problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (run-time errors)
- A program may run, but produce incorrect results, perhaps using an incorrect formula (logical errors)

Basic Program Development



Problem Solving

- The purpose of writing a program is to solve a problem
- Solving a problem consists of multiple activities:
  - Understand the problem
  - Design a solution
  - Consider alternatives and refine the solution
  - Implement the solution
  - Test the solution
- These activities are not purely linear – they overlap and interact

Problem Solving

- The key to designing a solution is breaking it down into manageable pieces
- When writing software, we design separate pieces that are responsible for certain parts of the solution
- An object-oriented approach lends itself to this kind of solution decomposition
- We will dissect our solutions into pieces called objects and classes

Object-Oriented Programming

- Java is an object-oriented programming language
- As the term implies, an object is a fundamental entity in a Java program
- Objects can be used effectively to represent real-world entities
- For instance, an object might represent a particular employee in a company
- Each employee object handles the processing and data management related to that employee

Objects

- An object has:
  - state - descriptive characteristics
  - behaviors - what it can do (or what can be done to it)
- The state of a bank account includes its account number and its current balance
- The behaviors associated with a bank account include the ability to make deposits and withdrawals
- Note that the behavior of an object might change its state

Classes

- An object is defined by a class
- A class is the blueprint of an object
- The class uses methods to define the behaviors of the object
- The class that contains the main method of a Java program represents the entire program
- A class represents a concept, and an object represents the embodiment of that concept
- Multiple objects can be created from the same class

Objects and Classes

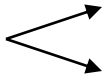
A class
(the concept)

An object
(the realization)

John's Bank Account
Balance: $5,257

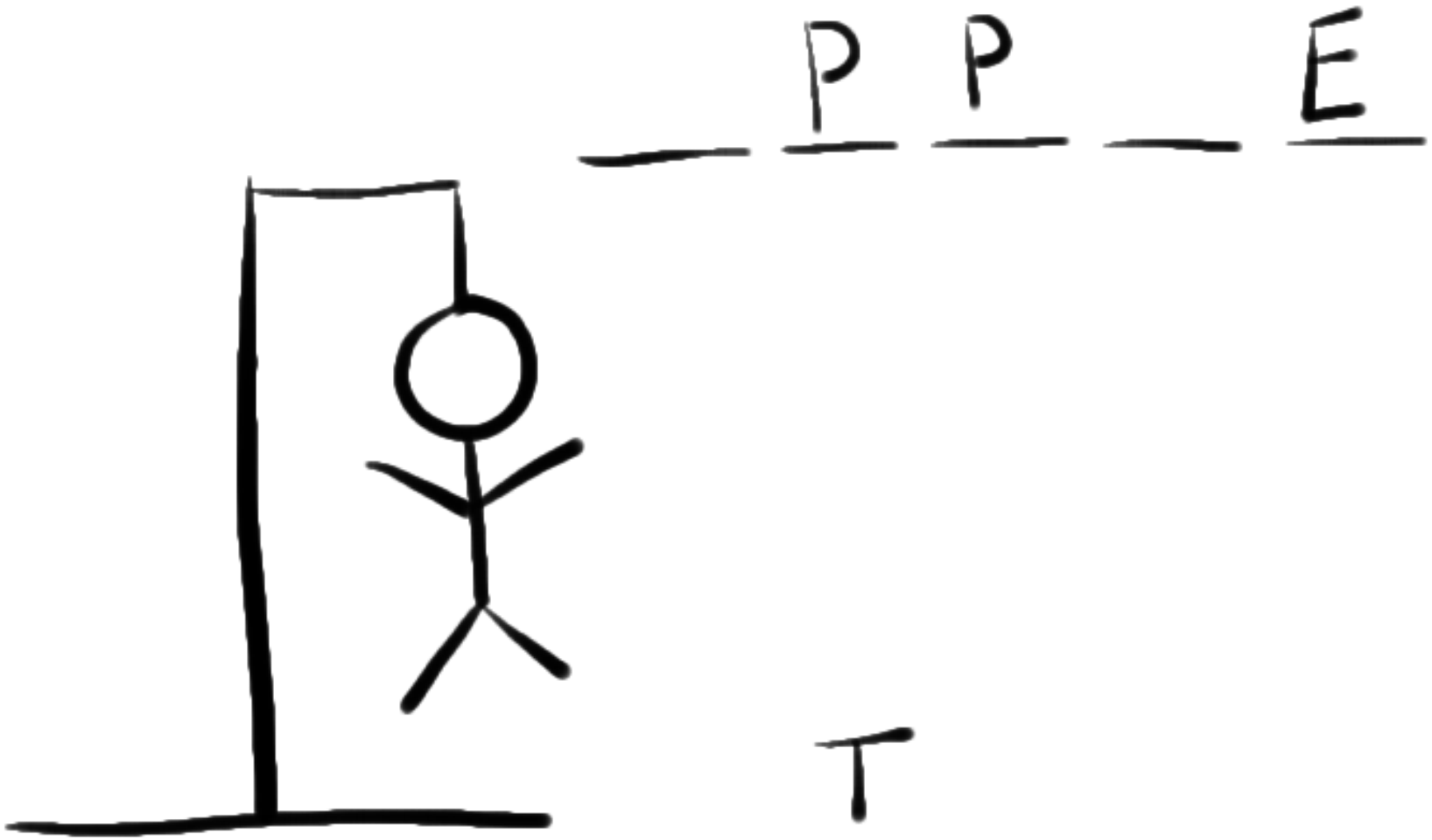
Bill's Bank Account
Balance: $1,245,069

Multiple objects
from the same class



Mary's Bank Account
Balance: $16,833

Exercise - Hangman



Introduction to Exercise

- Simple design exercise similar to what you will do for your CS102 project (and what happens in industry!)
- Stages: Requirements, UI, Detailed Design, implementation, Testing, Maintenance, ...
- Objective: Demonstrate process and problems associated with building software in groups
- Being aware of the problems may help you avoid some of them!

Requirements Stage

- Form groups of four or five
  - Later project groups can be different

- Come up with a written description of the game- explain it to little brother/sister!

Hangman - Description

- The player must find a word chosen secretly by the program
- The player can try one letter at a time and the program says how many times and where the letter appears in the secret word
- If the player tries more than a certain number of incorrect letters (i.e. letters which do not appear in the secret word), he/she loses the game and the program tells them the secret word
- Otherwise he continues until he uncovers the complete word

Hangman - Description

- Usually, players are able to see the partially formed word made by the letters so far correctly guessed, any unknown letters being left blank
- They may also be able to see the set of all letters that might be in the word (with those already used possibly being removed), as well as the number of incorrect tries made so far
  - This is often shown graphically, by the number of visible body parts of a cartoon character which is hung when complete -hence the name of the game!

User-Interface Stage

- Communicating the UI design... how?

- Set of pictures with "links" saying "if this happens" the result is "this”

- See the example…

Detailed Design Stage

- First design, then implement! But how do we come up with design?
- In groups again, have one person be the "display", one the "game" and one the "player" (other people can act as guides and recorders)
- The display needs certain info to produce the UI... it needs to ask the "game" (model) for this.
  - Exactly what does it ask for? What info, if any, does the game need to supply? What are the data types?
- User can see display, what do they need to ask the model to do? Types, etc. again.
- Updating the display may require additional info... and so on.
- The end result should be a good set of methods and properties for the game/model (and GUI/display) classes.

Design

- class Hangman
- constructors
  + Hangman()
    // default max 6 incorrect tries, English alphabet,
    // chooses secretWord from fixed list.
- properties
  - secretWord : StringBuffer
  - allLetters : StringBuffer
  - usedletters : StringBuffer
  - numberOfIncorrectTries : int
  - maxAllowedIncorrectTries : int
  - knownSoFar : StringBuffer // secretWord but with chars not yet found blanked out
- methods
  - + getAllLetters() : String
  - + getUsedLetters() : String
  - + getNumOfIncorrectTries() : int
  - + getMaxAllowedIncorrectTries : int
  - + getKnownSoFar() : String // returns partial word formed with known letters only
  - + tryThis(letter) : int // returns number of occurrences of letter in secretWord
  - + isGameOver() : boolean
  - + hasLost() : boolean
  - - chooseSecretWord() // initially use fixed list, called from constructor

Notes

- constructor:
  - set all letters to English alphabet, set max allowed incorrect tries to 6, no of incorrect tries to 0, used letters to empty set, secret word to result of calling choose secret word method, knowsofar to StringBuffer of same length as secret word, but all characters are stars ('*').
- tryThis(letter):
  - returns number of times the letter occurs in the secret word. Adds letter to used letters. Updates known so far to show the letter at each position it exists in secret word. If the letter is not in secret word then increment number of incorrect tries. Method should return error values to indicate if the letter is not valid (-1), if the letter was already used (-2), and indicate if game over (-3).
- chooseSecretWord:
  - returns a word chosen at random from a fixed array of words (defined in program code.) Also write another version of the method that reads the set of words from a text file and selects one from that.
- main:
  - play the game, by creating an instance of Hangman class and allowing the user to interact with it (in text mode using the keyboard only.) The program must repeatedly get letters from the user and try them...

Tasks

- Each group will code one of the below:

- main method - plays the game by creating instance of Hangman class & calling its methods.
- constructor for Hangman class
- tryThis method
- chooseSecretWord method
- hangman class (but with the bodies of the constructor, chooseSecretWord, & tryThis methods being empty.)

- Please email your Java code as an attachment to your instructor 24 hours before your next class, with subject line "CS102 - Hangman - SecNo - pieceNo".