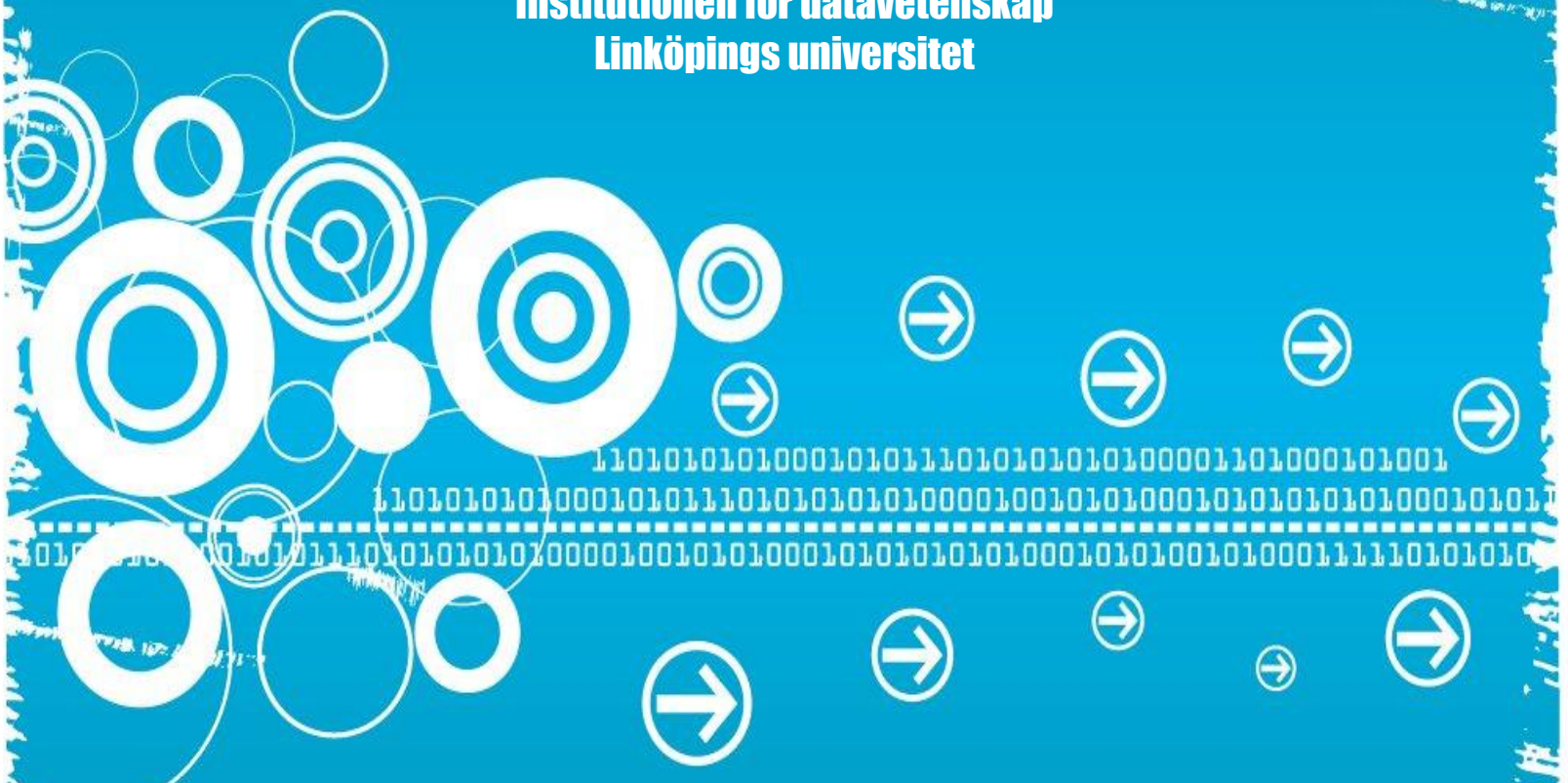
TDDC77 - Objektorienterad programmering

Laboration 5

Institutionen för datavetenskap
Linköpings universitet



5.1 Word Quiz*

I denna uppgift ska ni skriva ett glosprogram, dvs ett program som förhör användaren på glosor. Er uppgift är att skriva en klass, WordQuiz, som förhör användaren på ett antal termer som finns i en given ordlista. Användaren måste fortsätta förhöret ända tills alla ord har klarats. När alla termer översatts korrekt skriver programmet ut antalet gjorda försök.

```
import java.util.Scanner;

public class WordQuiz {
    private Scanner input = new Scanner(System.in);
    public WordQuiz(Dictionary dictionary) {
        // Er kod ska in här…
    }

    public void runQuiz() {
         // ...och här

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

      // Skapa en tom ordlista på ngt sätt och fyll den med ord. Dictionary
      sweng = ...;
      sweng.add("hej", "hello");
      sweng.add("hej", "hi");
      sweng.add("hej", "hey");
      sweng.add("godnatt", "good night");
      sweng.add("nattinatti", "good night");
      sweng.add("f˚agel", "bird");
      sweng.add("hund", "dog");
      sweng.add("katt", "cat");
      WordQuiz wq = new WordQuiz(sweng);
      wq.runQuiz();
    }
}
```

```
% java WordQuiz
fägel = bird
katt = katz
Fel!
hund = dogg
Fel!
hej = hi
godnatt = good nite
Fel!
nattinatti = good night
```

```
Du missade 3 ord.Jag förhör dig på dessa igen.
katt = catz
Fel!
hund = dog
godnatt = good night
Du missade 1 ord.Jag förhör dig på dessa igen.
katt = cat
Grattis, du klarade alla glosorna på 3 försök.
```

Ni kommer att behöva skriva ett antal hjälpklasser: `Word` och `Dictionary`. Dessa bör innehålla följande metoder:

```
/**
* Denna klass representerar ett ord, vilket består av en teckensekvens kallad
text.
*/
public class Word {

	/**
	* Skapar ett nytt ord med den givna texten.
	*/
	public Word(String text){}

	/**
	* Jämför detta ord med det specificerade objektet. Resultatet  är
	* true om och endast om obj också är ett ord (Word) och har
	* samma text.
	*/
	public boolean equals(Object obj){}


	/**
	* Returnerar hashkoden för detta ord beräknat på ordets text.
	*/
	public int hashCode(){}

	/**
	* Returnerar texten för detta ord.
	*/
	public String toString(){}

}
```

Vi ser att `Word` inte definierar några nya metoder utan endast åsidosätter tre ärvda metoder från klassen `Object`. Vi använder klassen `Word` istället för att använda `String`.

```
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Set;

/**
* Denna klass modellerar en ordlista (Dictionary). En ordlista
* associerar termer med betydelser. En term kan mappas till flera betydelser.
* Både term och betydelse representeras med klassen Word.
*/
public class Dictionary {

	/**
	* Lägger till termen t till ordlistan med betydelsen m. Om termen
	* redan  är tillagd med angiven betydelse händer ingenting.
	*/
	public void add(Word t, Word m){}

	/**
	* Bekvämare sätt att anropa add för 2 strängar  än
	* add(Word, Word).
	*/
	public void add(String t, String m){}

	/**
	* Returnerar en icke-null mängd med ordlistans alla termer.
	*/
	public Set<Word> terms(){}

	/**
	* Slår upp och returnerar en mängd av betydelser till t, eller
	* null om t inte finns i ordlistan.
	*/
	public Set<Word> lookup(Word t){}

	/**
	* Skapar och returnerar en ny ordlista på det motsatta språket, dvs, alla
	* betydelser blir termer och alla termer blir betydelser. T.ex. en
	* svensk-engelsk ordlista blir efter invertering engelsk-svensk.
	*/
	public Dictionary inverse();

	/**
```

```
        * Läser in orden från den givna strömmen och lägger dessa i ordlistan.
        */
        public void load(InputStream is);

        /**
        * Lagrar ordlistan på den givna strömmen.
        */
       public void save(OutputStream os);
}
```

Tänk på att om ordlistan ännu inte har några ord inlagda så ska ändå inte metoden `terms` returnera `null` utan en tom mängd/array. Skillnaden är bl.a. att en tom mängd ändå är en mängd som man kan använda t.ex. metoden `isEmpty`. Ett null-värde kan man inte göra någonting speciellt med.
Internt bör ordlistan använda sig av en `Map`-struktur (se `java.util`). Lämpligtvis mappar vi ord mot mängder av betydelser. Alltså `Map<Word, Set<Word>>`. De konkreta typer som kommer på fråga är alltså: HashMap och HashSet. Se vidare föreläsningarna. Ni ska på ett översiktligt sätt förstå hur dessa fungerar och vilka krav som ställs på nycklarna. (Detaljer kommer i TDDC91.)
Ni ska också skriva metoder för att spara och ladda ord till och från en ström. Här får ni inte använda er av Serialization, vilket är Javas egna API för att skriva och läsa datastrukturer. Tanken är att ni ska hitta på ett eget sätt för detta.

När programmet startas ska det fråga efter namnet på filen där ordlistan är lagrad.