Java enums

Enum types for fixed sets of constants

How do we create enumerable lists of constants?

We use enum types!

New keyword: `enum`

Enums work like classes and you declare one like this:

```
public enum WeekDay{

 // UPPER_CASE_CONSTANTS

}
```

Simple example

```
public enum Bill{
    TWENTY,FIFTY,HUNDRED,FIVE_HUNDRED,THOUSAND;
}
```

The above create an enumerable type for the Swedish bills (bank notes).

What makes Java enums special?

Enums can have constructors (private or package private):

```
public enum Bill{
  TWENTY(20),
  FIFTY(50),
  HUNDRED(100),
  FIVE_HUNDRED(500),
  THOUSAND(1000);
  private int value;
  Bill(int value){
    this.value = value;
  }
}
```

What makes Java enums special?

Enums have methods and can declare new ones.

The interesting inherited methods are:

name() - the name of the constant as declared

ordinal() - the ordinal of the constant (starts with 0)

What can we use methods for?

Accessor methods which allows for a range of values for each constant:

```
public enum WeekDay{
    SUNDAY("Söndag"),
    MONDAY("Måndag"),
    TUESDAY("Tisdag"),
    WEDNESDAY("Onsdag"),
    THURSDAY("Torsdag"),
    FRIDAY("Fredag"),
    SATURDAY("Lördag");
    private String swedish;
    private WeekDay(String swedish){
     this.swedish = swedish;
    }
    public String swedishName(){
        return swedish;
    }
....
```

Since enums declare constants we can use ==

```
public boolean isHoliday(){
  return this==SATURDAY||this==SUNDAY;
}
public boolean isWeekday(){
  return !isHoliday();
}
```

Enums also have some implicit methods

The compiler adds some implicit method. A useful one is values()

```
for(WeekDay day : WeekDay.values()){

  // Do stuff with day

}
```

Declaring a variable of enum type

```
WeekDay aDay = WeekDay.SATURDAY;
System.out.printf("The Swedish name for %s is %s\n",
                  aDay,
                  aDay.swedishName());
```

What about toString() ?

The default implementation of toString() returns the constant name as declared (e.g. MONDAY) as a String.

But we can of course override that behaviour:

```
@Override
public String toString(){
  return upperFirst(name());
}
private String upperFirst(String s){
  return s.charAt(0) + s.substring(1).toLowerCase();
}
```

Example enum for week days

```
public enum WeekDay{
    SUNDAY("Söndag"),
    MONDAY("Måndag"),
    TUESDAY("Tisdag"),
    WEDNESDAY("Onsdag"),
    THURSDAY("Torsdag"),
    FRIDAY("Fredag"),
    SATURDAY("Lördag");
    private String swedish;
    private WeekDay(String swedish){
        this.swedish = swedish;
    }
```

Example enum for week days

```
    public String swedishName(){
     return swedish;
    }
    public boolean isHoliday(){
     return this==SATURDAY||this==SUNDAY;
    }
    public boolean isWeekday(){
     return !isHoliday();
    }
    @Override
    public String toString(){
     return upperFirst(name());
    }
    private String upperFirst(String s){
     return s.charAt(0) + s.substring(1).toLowerCase();
    }
}
```

Main class

```
public class EnumTest{
  public static void main(String[] args){
    for(WeekDay day : WeekDay.values()){
      System.out.printf("%s is in Swedish: %s and is %s\n",
                        day,
                        day.swedishName(),
                        (day.isHoliday()?"a holiday"
                            :"a weekday."));
    }
    WeekDay aDay = WeekDay.SATURDAY;
    System.out.printf("The Swedish name for %s is %s\n", aDay, aDay.swedishName());
  }
}
```

Compile and run example (with package)

```
$ javac enumexample/EnumTest.java && java enumexample.EnumTest
Sunday is in Swedish: Söndag and is a holiday
Monday is in Swedish: Måndag and is a weekday.
Tuesday is in Swedish: Tisdag and is a weekday.
Wednesday is in Swedish: Onsdag and is a weekday.
Thursday is in Swedish: Torsdag and is a weekday.
Friday is in Swedish: Fredag and is a weekday.
Saturday is in Swedish: Lördag and is a holiday
The Swedish name for Saturday is Lördag
```