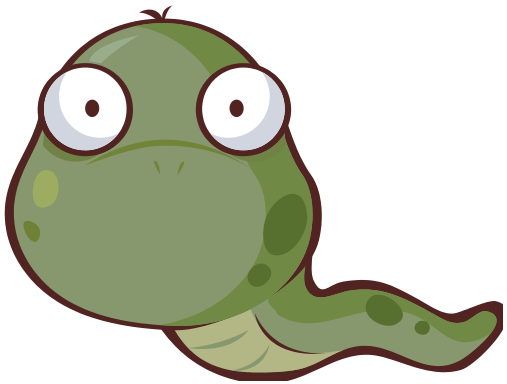
Of enumerations, switches and snakes



Overview

- Avoiding magic numbers
  - Variables takes on a small set of values
  - Use descriptive names instead of literal values
- Testing a variable against many values
  - Execute a block of code depending on the value
  - Avoid a big if-else-if-else if-… block
- Making a snake game

Variables from a set of values

- Magic numbers
  - Where did the value come from?
  - What does it mean?
  - What if you mistype the number?
  - What if you want to keep value in specific range?

```
int direction = 0;

...

if ((direction == 1) || (direction == 3) ||
    (direction == 5) || (direction == 7))
{ /* TBD */ }

direction = 0;       // Valid???
direction = 8;       // Valid???
direction = -2729;   // Valid???
```

- Solution 1: Create final constants
  - Descriptive names means everybody can read
  - Bugs less likely, typo in name = compile error
  - Final keyword ensures nobody can change value

```
final int NORTH      = 0;
final int NORTHEAST  = 1;
final int EAST       = 2;
final int SOUTHEAST  = 3;
final int SOUTH      = 4;
final int SOUTHWEST  = 5;
final int WEST       = 6;
final int NORTHWEST  = 7;

int direction = NORTH;

...

if ((direction == NORTHEAST) || (direction == SOUTHEAST) ||
    (direction == SOUTHWEST) || (direction == NORTHWEST))
{ // TBD }
```

Constants not always ideal

```
final int NORTH      = 0;
final int NORTHEAST  = 1;
final int EAST       = 2;
final int SOUTHEAST  = 3;
final int SOUTH      = 4;
final int SOUTHWEST  = 5;
final int WEST       = 6;
final int NORTHWEST  = 7;
```

Problem 1: Tedious to type. Also easy to mess up, e.g. setting two constants to same value.

```
int direction = 0;

...

if ((direction == NORTHEAST) || (direction == SOUTHEAST) ||
    (direction == SOUTHWEST) || (direction == NORTHWEST))
{/* TBD */}
```

Problem 2: Not forced to use, we can avoid using the friendly names.

```
direction = 0;      // Valid???
direction = 8;      // Valid???
direction = -2729;  // Valid???
```

Problem 3: Not forced to stay in range. What does it mean to be 8 or -2729 if you are a compass direction?

Enumerations

- A better solution: enumerations
  - Specifies exact set of friendly names
  - Compiler ensures we stay in range

```
enum Compass {NORTH, NORTHEAST, EAST, SOUTHEAST,
              SOUTH, SOUTHWEST, WEST, NORTHWEST}
public class CompassTest
{
  public static void main(String [] args)
  {
     Compass direction = Compass.NORTH;
     if ((direction == Compass.NORTHEAST) ||
         (direction == Compass.SOUTHEAST) ||
         (direction == Compass.SOUTHWEST) ||
         (direction == Compass.NORTHWEST))
        {/* TBD */}

     direction = 0;
  }
}
```

Easiest to declare outside class. Semicolon is optional.

Now a compile error.
Way to watch our back compiler!

Enumeration tricks

- Enumerations
  - Actually objects with a few handy methods:

`toString()`	Print out friendly name corresponding to value of variable
`values()`	Returns array of all the possible values type can take on

```
enum Compass {NORTH, NORTHEAST, EAST, SOUTHEAST,
              SOUTH, SOUTHWEST, WEST, NORTHWEST}

...
for (Compass d : direction.values())
{
   if (checkMonster(hero, d))
      System.out.println("You see a monster to the " +
                         d.toString());
}
```

Conditional action from a set

- Do something depending on a value value
  - if-else if-else if... statements can get tedious

```
if (day == 1)
   monthStr = "Monday";
else if (day == 2)
   monthStr = "Tuesday";
else if (day == 3)
   monthStr = "Wednesday";
else if (day == 4)
   monthStr = "Thursday";
else if (day == 5)
   monthStr = "Friday";
else if (day == 6)
   monthStr = "Saturday";
else if (day == 7)
   monthStr = "Sunday";
else
   monthStr = "Invalid day!";
```

Set a String variable `monthStr` to a string according to the integer value in the `day` variable.

Conditional action from a set

- switch statement
  - Works with: `byte, short, char, int, enumerations`

```
switch (day)
{
   case 1:  monthStr = "Monday";       break;
   case 2:  monthStr = "Tuesday";      break;
   case 3:  monthStr = "Wednesday";    break;
   case 4:  monthStr = "Thursday";     break;
   case 5:  monthStr = "Friday";       break;
   case 6:  monthStr = "Saturday";     break;
   case 7:  monthStr = "Sunday";       break;
  default: monthStr = "Invalid day!"; break;
}
```

case block normally ends with a break

default block is optional, but if present executes if no other case matched. Like the else in an if-else if-else statement.

switch statement

```
Compass direction = Compass.NORTH;

switch (direction)
{
    case NORTH:
        hero.move(0, 1);
        System.out.println("Walking north");
        break;
    case SOUTH:
        hero.move(0, -1);
        System.out.println("Walking south");
        break;
    case EAST:
        hero.move(1, 0);
        System.out.println("Walking east");
        break;
    case WEST:
        hero.move(-1, 0);
        System.out.println("Walking west");
        break;
}
```

Note: normally you need "Compass.", but not in switch case since Java knows type

You can have as many statements as you want between case and break.

Buggy switch statement

```
Compass direction = Compass.NORTH;

switch (direction)
{
   case NORTH:
      hero.move(0, 1);
      System.out.println("Walking north");
   case SOUTH:
      hero.move(0, -1);
      System.out.println("Walking south");
   case EAST:
      hero.move(1, 0);
      System.out.println("Walking east");
   case WEST:
      hero.move(-1, 0);
      System.out.println("Walking west");
}
```

case blocks with fall through to next block if you don't use the break statement!

```
Output:

Walking north
Walking south
Walking east
Walking west
```

Falling through cases

```
Compass direction = Compass.SOUTHEAST;

switch (direction)
{
    case NORTHWEST:
    case NORTHEAST:
    case NORTH:
        System.out.println("Heading northbound");
        break;
    case SOUTHWEST:
    case SOUTHEAST:
    case SOUTH:
        System.out.println("Heading southbound");
        break;
}
```

Sometimes falling through to next case block is what you want.

Easy way to do same thing for a set of discrete values.

```
Output:

Heading southbound
```

Snake game