COMP 110/L Lecture 10

Kyle Dewey

Outline

- switch

switch

Problem

`if` is verbose when checking many conditions.

Problem

`if` is verbose when checking many conditions.

```
if (x == 5) {
  return “foo”;
} else if (x == 6) {
  return “bar”;
} else if (x == 7) {
  return “baz”;
} else if (x == 8) {
  return “blah”;
} else {
  return “unknown”;
}
```

Enter `switch`

`switch` allows for multiple == conditions to be checked

```
if (x == 5) {
  return “foo”;
} else if (x == 6) {
  return “bar”;
} else if (x == 7) {
  return “baz”;
} else if (x == 8) {
  return “blah”;
} else {
  return “unknown”;
}
```

Enter `switch`

`switch` allows for multiple == conditions to be checked

```
if (x == 5) {
  return “foo”;
} else if (x == 6) {
  return “bar”;
} else if (x == 7) {
  return “baz”;
} else if (x == 8) {
  return “blah”;
} else {
  return “unknown”;
}
```

```
switch (x) {
case 5:
  return “foo”;
case 6:
  return “bar”;
case 7:
  return “baz”;
case 8:
  return “blah”;
default:
  return “unknown”;
}
```

Example:

SwitchBasic.java

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (x) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (1) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

-If the value we switch on is 1...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (1) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

-...then jump to case 1...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (1) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

-...and start executing statements from this point.

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (1) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

-In this case, because it's a return, execution stops here (returning to whoever called this)

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (3) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

-If the value we switch on is 3...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (3) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
  }
```

-...then we jump to the default case, as there is no case for 3

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (3) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
  }
```

–We would then print out "huh"...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (3) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
}
```

-...and then simply trail out of the switch statement
-Whichever statement follows the switch would be executed, just as with if

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (2) {
case 1:
  return "hi";
case 2:
  System.out.println("bye");
default:
  System.out.println("huh");
}
```

-If the value we switch on is 2...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (2) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
  }
```

-...then we jump to the case for 2...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (2) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
  }
```

-...and then start executing subsequent statements.
-We'd first print "bye"...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (2) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
  }
```

-...but because nothing stopped us, we'd go to the next statement.
-In this case, this would mean we'd also print "huh"...

`switch` Semantics

- Look at the thing you're `switch`ing on
- Jump to the applicable case
- Keep running statements until something stops you

```
switch (2) {
  case 1:
    return "hi";
  case 2:
    System.out.println("bye");
  default:
    System.out.println("huh");
}
```

-...and then would trail out of the switch, just as before

Example:

SwitchFallthrough.java

Preventing “fall-through”

The `break` statement will exit out of a `switch`.

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (x) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
default:
   System.out.println("huh");
}
```

-If I take the switch from before...

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (x) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
   break;
default:
   System.out.println("huh");
}
```

-...and then throw a break in...

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (2) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
   break;
default:
   System.out.println("huh");
}
```

-...this now behaves differently on case 2

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (2) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
   break;
default:
   System.out.println("huh");
}
```

-We'd still jump to the case 2...

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (2) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
   break;
default:
   System.out.println("huh");
}
```

-We'd still execute the subsequent statement (printing "bye")...

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (2) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
   break;
default:
   System.out.println("huh");
}
```

-...but when we reach the break, we exit out of the switch

Preventing "fall-through"

The `break` statement will exit out of a `switch`.

```
switch (2) {
case 1:
   return "hi";
case 2:
   System.out.println("bye");
   break;
default:
   System.out.println("huh");
}
```

-...but when we reach the break, we exit out of the switch
-End result: "bye" is printed, but not "huh"

Example:

SwitchBreak.java

`switch` and Testing

Each `case` is a test candidate, as is `default`.

`switch` and Testing

Each `case` is a test candidate, as is `default`.

```
int result = 0;
switch (input) {
  case 1:
    result = result + 2;
  case 2:
    result = result + 5;
  default:
    result = result + 12;
  }
```

`switch` and Testing

Each `case` is a test candidate, as is `default`.

```
      int result = 0;
      switch (input) {
1     case 1:
        result = result + 2;
      case 2:
        result = result + 5;
      default:
        result = result + 12;
      }
```

`switch` and Testing

Each `case` is a test candidate, as is `default`.

```
      int result = 0;
      switch (input) {
1       case 1:
          result = result + 2;
2       case 2:
          result = result + 5;
        default:
          result = result + 12;
        }
```

`switch` and Testing

Each `case` is a test candidate, as is `default`.

```
      int result = 0;
      switch (input) {
1       case 1:
          result = result + 2;
2       case 2:
          result = result + 5;
3       default:
          result = result + 12;
      }
```