Relational operators


```
if (<condition>) {...}
else if (<condition>) {...}
```

What are these conditions? $\Rightarrow$ *boolean* conditions (true/false)

Use: `==`, `!=`, `<`, `<=`, `>`, `>=`, `string.equals()`

NOT: `x = y` or `x =< y`

booleans

Also can have `boolean`-type variables/literals:

```
boolean agree = (x == y);
if(agree) {...}
agree = true;
```

Practice

Let's write a dice rolling program!

1. Prompt user for number of sides on the die
2. Get user input
3. Generate random number as a roll
4. Output result of roll
5. Output "even" if the roll is even, "odd" if the roll is odd

How to think about structure

Let's design this program before we write the code:

1. Decide on the branching condition
2. What needs to happen when the condition is true?
3. Does anything need to happen when it isn't? What?
4. Check the relational operators
5. Remove any duplicate stuff
6. Test both branches
7. NOW write the code.

Nested if statements

We can put if statements inside of if statements.

```
if(status.equals("Married")) {
    if (income <= 32000) { tax = RATE1 * income; }
    else {
     tax1 = RATE1 * 32000;
     tax2 = RATE2 * (income - 32000);
    }
}
else {
    if (income <= 64000) { tax = RATE1 * income; }
    else {
     tax1 = RATE1 * 64000;
     tax2 = RATE2 * (income - 64000);
    }
}
```

Code Tracing

If there's only one line of code after an `if`, you don't *need* curly braces, but they're still a good idea.

Why? $\rightarrow$ Handout!

switch statements

When you have many choices with a simple variable type (e.g. int, char, etc), you can make things easier with a switch statement instead of many if-else-ifs:

```
int digit = /* something */;
switch (digit) {
    case 1: digitName = "one"; break;
    case 2: digitName = "two"; break;
    case 3: digitName = "three"; break;
    case 4: digitName = "four"; break;
    default: digitName = ""; break;
}
```

Always have the break; on the default case (style). Has a function elsewhere, though!