Strings

- `pat` = `aa`
- `tar` = `a`
- `verbose` = `true`

- `match(0, 0)`
  - `find` = `true`
  - `replaceAll` = `true`

- `match(1, 1)`
  - `pat` = `a`
  - `tar` = `a`
  - `true`

- `match(0, 0)`
  - `return true`

- `atEnd` method:
  - `atEnd(2, tar) -> return true`
  - `atEnd(0, tar) -> return false`
  - `atEnd(1, tar) -> return false`

- `match`:
  - `Pat: aa`
  - `Tar: aa`

- `1st activation`:
  - `return true`

- `2nd activation`:
  - `return true`

- `Empty string`:
  - `return true`
Overloaded method names
example:
```java
match (p)
```
overloaded:
```java
match (int pInd, int tInd);
```
method
```java
for var methods
name ⇒ Some name
different signature
Different bodies!
```

The one that is called is the one whose signature matches
the type of the parameters coded in the call!

It's convenient 😊

```java
match () → tries to match the whole pattern with
return match(0, 0);
```
```java
match (int pInd, int tInd)
```
```java
tries to match pattern from pInd to end with target from tInd to end.
```
A tic-tac-toe playing program can use backtracking search to play an optimum game!