```
class NamedPony {
  constructor(public name: string, private speed: number) {
  }

  run() {
    logger.log(`pony runs at ${this.speed}m/s`);
  }
}
    const pony = new NamedPony('Rainbow Dash', 10);
    // defines a public property name with 'Rainbow Dash'
    // and a private one speed with 10
```

Which is the same as the more verbose:

```
class NamedPonyWithoutShortcut {
  public name: string;
  private speed: number;

  constructor(name: string, speed: number) {
    this.name = name;
    this.speed = speed;
  }

  run() {
    logger.log(`pony runs at ${this.speed}m/s`);
  }
}
```

These shortcuts are really useful and we'll rely on them a lot in Angular!

5.8. Working with other libraries

When working with external libraries written in JS, you may think we are doomed because we don't know what types of parameter the function in that library will expect. That's one of the cool things with the TypeScript community: its members have defined interfaces for the types and functions exposed by the popular JavaScript libraries!

The files containing these interfaces have a special `.d.ts` extension. They contain a list of the library's public functions. A good place to look for these files is DefinitelyTyped. For example, if you want to use TS in your AngularJS 1.x apps, you can download the proper file from the repo directly with NPM:

```
npm install --save-dev @types/angular
```

or download it manually. Then include the file at the top of your code, and enjoy the compilation checks: