Noroff
School of technology
and digital media

Monday - Week 2

Building a .jar

Building a .jar file (with the command line)

- This is an extra step after compilation
- It places the byte code into a single package

Source code (.java) —[javac]—> Byte code (.class)

Byte code [.class] —[jar]—> JAR file (.jar)

Why do this?

- The JAR file gives us the following:
  - Compression
  - Portability
  - Versioning
  - Security

Follow along steps!

Part 1

- Start with source code!
- Build a “Hello world program”
- Call it program.java

Part 2

- Compile it:
  `javac program.java`

- Now we have a program.class

Part 3

- Build the JAR file:

```
jar cf program.jar program.class
```

Step 4

- Run it!

```
java -jar program.jar
```

no main manifest attribute, in program.jar

Step 5

- Rebuild with main entry point defined!

```
jar cfe program.jar program program.class
```

Step 6

- Run it again

```
java -jar program.jar
```

Does it work?

- I found it overly complicated to get additional .class files included
- Ideally you should create a “manifest file”

- We will be switching to IntelliJ to help manage these steps

Noroff
School of technology
and digital media

Monday - Week 2

Project Structure & Packages

Structure Recap

- Working with the structure in command line is convoluted
- We are forced to manually match filenames and folders with packages

Problems

- We “should” be stricter than we have been
- Single source applications tend to bypass some issues
    - But we cannot build meaningful solutions in a single file

Let switch to IntelliJ

Noroff
School of technology
and digital media

Monday - Week 2

Tasks

Task 13 - Part 1

- You may do this in IntelliJ
- Build a an Animal Class
- An animal can be Herbivore, Carnivore or Omnivore
- Create Interfaces for movement types:
    - Walk, Run, Fly, Swim Climb
    - Each should write a string the the console

Task 13 - Part 2

- The animal class should provide a means for “exposing” a list of movement types (Perhaps a list of enums?)
- Build a console application that that creates a collection of Animals
- Create a variety of animals (Min. 3)
- For each animal, randomly pick one of its movement types and display (run it).

Task 13 - Part 3

- Example output for cat (walk, run, climb):

```
> The cat can walk, run, and climb.
> The cat runs.
```

Task 13 - Bonus Option

- Switch up the structure,
  - Pre-compile the animal class into a package
  - Import the package separately into a different project and use that instead of the uncompiled version