Java's Built-in Data Definitions

- **int**
  
  ```
  1   5999   -10
  ```

- **double**
  
  ```
  1.1  5999.33  -10.01
  ```

- **boolean**
  
  ```
  true   false
  ```

- **String**
  
  ```
  "hello"  "See you later!"
  ```
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

```java
class Snake {
    String name;
    double weight;
    String food;

    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}
```
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Next is the name for the data definition; by convention, the name is capitalized
Compound Data in Java

Beginner Scheme:

; A snake is
;  (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

```java
class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}
```
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

For each part of the compound value, write type then name then ;, one line for each part; this is a field
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
  String name;
  double weight;
  String food;

  Snake(String name, double weight, String food) {
    this.name = name;
    this.weight = weight;
    this.food = food;
  }
}

After the parts, write the defined name again; this starts the constructor
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Then a (}
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Write each field again, but this time separate with , — these are the constructor arguments
Compound Data in Java

Beginner Scheme:

```
; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))
```

Beginner Java:

```java
class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}
```
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Then a {
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Each field, one more time...
this then .
then name then
= then name
then ;
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Closing } for the constructor
Compound Data in Java

Beginner Scheme:

; A snake is
; (make-snake sym num sym)
(define-struct snake (name weight food))

Beginner Java:

class Snake {
    String name;
    double weight;
    String food;
    Snake(String name, double weight, String food) {
        this.name = name;
        this.weight = weight;
        this.food = food;
    }
}

Closing } for the class declaration
Instances of Compound Data Types

Beginner Scheme:

```
(make-snake 'Slinky 12 'rats)
(make-snake 'Slimey 5 'grass)
```

Beginner Java:

```
new Snake("Slinky", 12, "rats")
new Snake("Slimey", 5, "grass")
```
Instances of Compound Data Types

Beginner Scheme:

(make-snake 'Slinky 12 'rats)
(make-snake 'Slimey 5 'grass)

Beginner Java:

new Snake("Slinky", 12, "rats")
new Snake("Slimey", 5, "grass")

new starts an instance (a value) of a class
Instances of Compound Data Types

Beginner Scheme:

```
(make-snake 'Slinky 12 'rats)
(make-snake 'Slimey 5 'grass)
```

Beginner Java:

```
new Snake("Slinky", 12, "rats")
new Snake("Slimey", 5, "grass")
```
Instances of Compound Data Types

Beginner Scheme:

\[
\text{(make-snake 'Slinky 12 'rats)} \\
\text{(make-snake 'Slimey 5 'grass)}
\]

Beginner Java:

\[
\text{new Snake("Slinky", 12, "rats")} \\
\text{new Snake("Slimey", 5, "grass")}
\]
Instances of Compound Data Types

Beginner Scheme:

(make-snake 'Slinky 12 'rats)
(make-snake 'Slimey 5 'grass)

Beginner Java:

new Snake("Slinky", 12, "rats")
new Snake("Slimey", 5, "grass")

Then field values separated by ,
Instances of Compound Data Types

Beginner Scheme:

(make-snake 'Slinky 12 'rats)
(make-snake 'Slimey 5 'grass)

Beginner Java:

new Snake("Slinky", 12, "rats")
new Snake("Slimey", 5, "grass")

Then )
class Dillo {
    double weight;
    boolean alive;
    Dillo(double weight, boolean alive) {
        this.weight = weight;
        this.alive = alive;
    }
}

new Dillo(2, true)
new Dillo(3, false)
class Posn {
    int x;
    int y;
    Posn(int x, int y) {
        this.x = x;
        this.y = y;
    }
}

new Posn(0, 0)
new Posn(1, -2)
class Ant {
    double weight;
    Posn loc;
    Ant(double weight, Posn loc) {
        this.weight = weight;
        this.loc = loc;
    }
}

new Ant(0.0001, new Posn(0, 0))
new Ant(0.0002, new Posn(1, -2))
Data with Variants

Beginner Scheme:

; An animal is either
;   - snake
;   - dillo
;   - ant

Beginner Java:

abstract class Animal {
}

class Snake extends Animal {
    ... as before ...
}
class Dillo extends Animal {
    ... as before ...
}
class Ant extends Animal {
    ... as before ...
}
Data with Variants

Beginner Scheme:
; An animal is either
;   - snake
;   - dillo
;   - ant

Beginner Java:
abstract class Animal {

abstract class for a data
definition with
variants

class Snake extends Animal {
    ... as before ...
}

class Dillo extends Animal {
    ... as before ...
}

class Ant extends Animal {
    ... as before ...
}
Data with Variants

Beginner Scheme:

; An animal is either
;   - snake
;   - dillo
;   - ant

Beginner Java:

abstract class Animal {
}

class Snake extends Animal {
  ...
}

class Dillo extends Animal {
  ...
}

class Ant extends Animal {
  ...
}

No fields and no constructor when a class merely groups variants
Data with Variants

Beginner Scheme:

; An animal is either
;   - snake;   - dillo;   - ant

Beginner Java:

```java
abstract class Animal {
}

class Snake extends Animal {
    ... as before ...
}

class Dillo extends Animal {
    ... as before ...
}

class Ant extends Animal {
    ... as before ...
}
```

Change the class for each variant by adding `extends` then the grouping class name, all before `{`
Data with Variants

Beginner Scheme: 

; An animal is either
;   - snake
;   - dillo
;   - ant

Beginner Java: 

abstract class Animal {
}

class Snake extends Animal {
    ... as before ...
}

class Dillo extends Animal {
    ... as before ...
}

class Ant extends Animal {
    ... as before ...
}
Variants in Java

• A data definition with variants must refer only to other data definitions (which are not built in)

; A grade is either
;    - false
;    - num

⇒

; A grade is either
;    - no-grade
;    - num-grade

; A no-grade is
;    (make-no-grade)
(define-struct no-grade ())

; A num-grade is
;    (make-num-grade num)
(define-struct num-grade (n))

• A data definition can be a variant in at most one other data definition