# Java Training

Day 7: Accessors, Constructors and the remaining Primitives

Download today’s slides:
go/java+espresso-training/day7

## Back to Bob’s Constructor

It’s convenient to set the state of an instance when it’s created

```
public class Constructor {
        public static void main(String[] args){
                Bob bob1 = new Bob(64);
                System.out.println(bob1.age);
        }
}

class Bob {
        int age;
        public Bob(int age){
                this.age = age;
        }
}
```

A **constructor** can have as many **arguments** as you like and you may have as many constructors as you have **unique argument lists**

```
> javac Constructor.java
> java Constructor

64
```

## *Deluxe* `Bob`, with **Constructor** and `toString()` override

```
public class Constructor {
        public static void main(String[] args){
                Bob bob1 = new Bob(64);
                // bob1.age = 22;   - compile error
                System.out.println(bob1);
        }
}

class Bob {
        private int age;   // access limited to Bob methods
        public String toString(){
                return "Bob: "+age+" yrs";
        }
        public Bob(int age){
                this.age = age;
        }
}
```

```
> javac Constructor.java
> java Constructor

Bob: 64 yrs
```

How do you get a Bob's age as a simple `int`?

## Accessors for Bob’s age

```
public class Constructor {
        public static void main(String[] args){
                Bob bob1 = new Bob(64);
                System.out.println(“Age of bob1=“+bob1.getAge());
        }
}

class Bob {
        private int age;  // access limited to Bob methods
        public Bob(int age){
                this.age = age;
        }

        public int getAge(){
                return this.age;
        }

}
```

This type of *Accessor* is called a **getter.** A **private** member variable and only a **getter** gives you *read-only* data

```
> javac Constructor.java
> java Constructor

Age of bob1=64
```

There are also **setter** *Accessors*.

But then what’s the point of *private*?

## Code Exercise: Add Bob’s height, Exercise #1

Add a `height` member variable to `Bob`. It should be an `int` and `private`. Then add a **getter** for `height` and modify the **constructor** so that Bobs can be initialized with both `age` and `height`. Also, update `main()` so that these new features are exercised.

```
class Bob {
        private int age;  // access limited to Bob methods
        public Bob(int age){
                this.age = age;
        }

        public int getAge(){
                return this.age;
        }

}
```

## Exercise #1, Possible Solution

```java
public class Constructor {
    public static void main(String[] args){
        Bob bob1 = new Bob(64, 33);
        System.out.println("Age of bob1=" + bob1.getAge());
        System.out.println("Height of bob1=" + bob1.getHeight());
    }
}

class Bob {
    private int age;
    private int height;

    public Bob(int a, int h){
        this.age = a;
        this.height = h;
    }
    public int getAge(){
        return this.age;
    }
    public int getHeight(){
        return this.height;
    }
}
```

## A *setter* for Bob’s age (including validation)

```
public void setAge(int a){
    if ((a < 0) || (a > 115)){
        System.out.println("Bad Bob age:" + a);
    } else {
        this.age = a;
    }
}
```

If you try:

```
bob1.setAge(400)
```

You’ll get and error and `bob1`’s age won’t be corrupted

## Exercise #2, add a **setter** for `height`.

## Using `setAge()` as a model, have the setter *validate* the `height` value.

```
public void setAge(int a){
    if ((a < 0) || (a > 115)){
        System.out.println("Bad Bob age:" + a);
    } else {
        this.age = a;
    }
}
```

*21 inches* a reasonable minimum `height`
*300 inches* is a safe human maximum `height`

If the `height` passed to the **setter** is out of bounds, *silently* discard it (no error message). Test the **setter** using your existing `main()`

## Exercise #2, Possible Solution

```
public void setHeight(int h){
    if ((h > 20) && (h < 301)){
        this.height = h
    }
}
```

*Silent failure* simplifies the logic a little as compared to `setAge()`

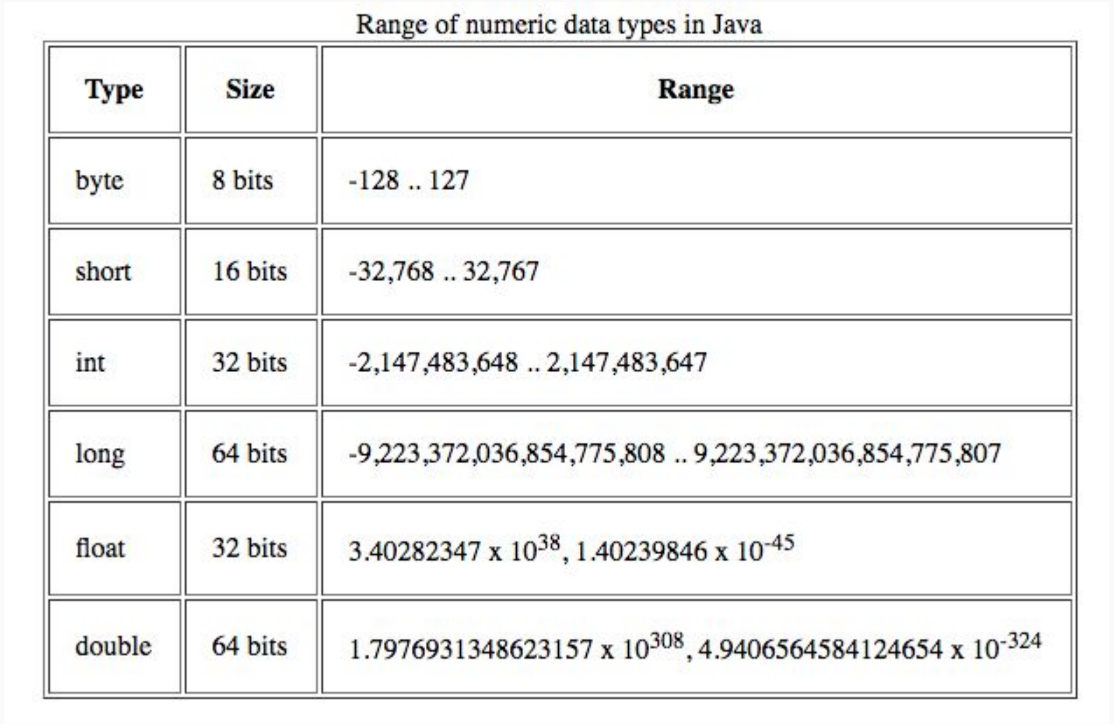
```
public void setAge(int a){
    if ((a < 0) || (a > 115)){
        System.out.println("Bad Bob age:" + a);
    } else {
        this.age = a;
    }
}
```

# The rest of the Numeric Java Primitives + Exercise #3

Examples with `double`:

```
double x = 45.4;
double y = x + 22.8;
Double  =
Double.valueOf(“98.6”);
```

The **boxed-type** for `double` is `Double`



Range of numeric data types in Java

| Type | Size | Range |
|---|---|---|
| byte | 8 bits | -128 .. 127 |
| short | 16 bits | -32,768 .. 32,767 |
| int | 32 bits | -2,147,483,648 .. 2,147,483,647 |
| long | 64 bits | -9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807 |
| float | 32 bits | $3.40282347 \times 10^{38}$, $1.40239846 \times 10^{-45}$ |
| double | 64 bits | $1.7976931348623157 \times 10^{308}$, $4.940656458412465 4 \times 10^{-324}$ |

Update `Bob` so that `height` variable is a `double` rather than an `int`
You’ll need to update the **member variable**, the **constructor** and the **accessors** for `height`

## Exercise #3, solution

```
class Bob {
    private int age;
    private double height;

    public Bob(int a, double h){
        this.age = a;
        this.height = h;
    }
    public double getHeight(){
        return this.height;
    }
    public void setHeight(double h){
        if ((h > 20.0) && (h < 301.0)){
            this.height = h;
        }
    }
}
```