Lecture mainly follows Sun's Java Tutorial available at
http://java.sun.com/docs/books/tutorial/
Java as a Programming Language

Programming Languages

- Imperative (“HOW”) vs. Declarative (“WHAT”)
  - Imperative
    - Procedural
    - Object-Oriented
  - Declarative
    - Logic-Based
    - Functional

Compiled Languages: (e.g. C)

Virtual Machine Languages: (e.g. Java)

Java as a Programming Language

Imperative Programming

- Imperative program: Sequence of statements
- Instructions change state (especially memory) of computer system

```java
boolean plato;
text horst;
text heiner;
text fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

```
memory (simplified model)
cell nr | cell name | cell content
--|---|---
1123 | plato | false
1124 |
1125 | horst | 101
1126 | heiner | 0
1127 |
1128 | fritz | 0
```

Imperative Programming

- Imperative program: Sequence of statements
- Instructions change state (especially memory) of computer system

```java
boolean plato;
text horst;
text heiner;
text fritz;
plato = false;
horst = 101;
heiner = 2;
fritz = horst + heiner;
horst = 2000;
```

```
memory (simplified model)
cell nr | cell name | cell content
--|---|---
1123 | plato | false
1124 |
1125 | horst | 2000
1126 | heiner | 2
1127 |
1128 | fritz | 103
```
Procedural Programming

• Group sequences of instructions into named "procedures" ("functions", "methods", "sub-routines" etc.)

```
int doSelfSumSquare(int someNumber) {
    int a;
    a = someNumber + someNumber;
    a = a * a;
    return a;
}
```

• Advantages
  • no copying of instruction sequences
  • better testing
  • modularity (e.g. code change inside procedure doesn't affect caller)
  • code re-use
  • etc.

In the example: Control flow is transferred to procedure, back to main program, back to procedure and back to main program

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;
```

```
int doSelfSumSquare(int someNumber) {
    int a;
    a = someNumber + someNumber;
    a = a * a;
    return a;
}
```

```
int horst;
int heiner;
horst = 101;
heiner = 2;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;
```

```
int doSelfSumSquare(int someNumber) {
    int a;
    a = someNumber + someNumber;
    a = a * a;
    return a;
}
```

In the example: Control flow is transferred to procedure, back to main program, back to procedure and back to main program
In the example: Control flow is transferred to procedure, back to main program, back to procedure and back to main program.
Java as a Programming Language

int horst;
int heiner;
horst = 101;
heiner = 27;
heiner = doSelfSumSquare(horst);
heiner = doSelfSumSquare(117);
horst = horst + 2;

int doSelfSumSquare(int someNumber) {
    int a;
    a = someNumber + someNumber;
    a = a * a;
    return a;
}

- In the example: Control flow is transferred to procedure, back to main program, back to procedure and back to main program

- Procedures often access global variables (bad style!)
- Goal: „Keep things local“ (better testing, code re-use etc.)
Object-oriented Programming:

- Object-oriented programming:
  - Group data and procedures into objects
  - Models of state and behaviour of real world objects
  - state "fields" ; behaviour "methods"

- Methods should mainly act on an object’s fields
- Classes: blueprints for objects → Objects: Instances of classes
- Advantages
  - Intuitive models
  - Information hiding
  - Increased modularity, locality etc.
  - Increased code re-use
  - etc.

```java
class Bicycle {
    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence(int newValue) {
        cadence = newValue;
    }

    void changeGear(int newValue) {
        gear = newValue;
    }

    void speedUp(int increment) {
        speed = speed + increment;
    }

    void applyBrakes(int decrement) {
        speed = speed - decrement;
    }
}
```

Source: [JTutorial]

```java
class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on these objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);
        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeCadence(3);

        void changeCadence(int newValue) {
            cadence = newValue;
        }

        void changeGear(int newValue) {
            gear = newValue;
        }

        void speedUp(int increment) {
            speed = speed + increment;
        }

        void applyBrakes(int decrement) {
            speed = speed - decrement;
        }
    }
}
```

Source: [JTutorial]
Java as a Programming Language

class BicycleDemo {
    public static void main(String[] args) {
        // Create two different Bicycle objects
        Bicycle bike1 = new Bicycle();
        Bicycle bike2 = new Bicycle();

        // Invoke methods on these objects
        bike1.changeCadence(50);
        bike1.speedUp(10);
        bike1.changeGear(2);

        bike2.changeCadence(50);
        bike2.speedUp(10);
        bike2.changeGear(2);
        bike2.changeCadence(40);
        bike2.speedUp(10);
        bike2.changeGear(3);
    }
}

Source: [JTutorial]
Java as a Programming Language

Example:

Interfaces

- Mammal
  - height: int
  - weight: int
  - eatSomething()

- Whale
  - height: int
  - weight: int
  - eatSomething()
  - dive()

- KartDive
  - dive()

- Vehicle
  - height: int
  - weight: int
  - speed: int
  - accelerate()
  - decelerate()
  - crash()

- Submarine
  - height: int
  - weight: int
  - speed: int
  - accelerate()
  - decelerate()
  - crash()
  - dive()

2 Language Basics

Deepening readings:

- http://java.sun.com/docs/books/tutorial/java/nutsandbolts/if.html
- http://java.sun.com/docs/books/tutorial/java/nutsandbolts/while.html

Language Basics – Variables

Variables

- Variables have a type
  - Primitive type
  - Reference type

<table>
<thead>
<tr>
<th>Type</th>
<th>Definition</th>
<th>Declaration</th>
<th>Instantiation</th>
<th>Manipulation</th>
<th>Equality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive</td>
<td>predefined</td>
<td>int a;</td>
<td>a = 117;</td>
<td>a = b + 42;</td>
<td>a == b;</td>
</tr>
<tr>
<td>Reference</td>
<td>class Student { // Fields and // methods ... }</td>
<td>Student heiner;</td>
<td>heiner = new Student();</td>
<td>heiner.yawn();</td>
<td>heiner.equals( sabine );</td>
</tr>
</tbody>
</table>
Language Basics – Variables

Variables have a type

- **Primitive type**
  ```java
  int horst = 101;
  long heiner;
  heiner = 235638465837465845;
  ```

- **Reference type**
  ```java
  Bicycle bike1 = new Bicycle();
  bike1.cadence = 0;
  long heiner;
  heiner = 235638465837465845;
  ```

Primitives Types

- **Primitive types (numeric):**
  - byte 8 bit
  - short 16 bit
  - int 32 bit
  - long 64 bit
  - float 32 bit
  - double 64 bit

Examples:

- `byte flags = 63;`
- `short bbb = 10133;`
- `int heiner = 234103234;`
- `long dong = -83628735682345;`
- `float fff = 5464.00345;`
- `float ggg = -345545.34534E-12f;`
- `double sss = 3245343455.555867;`
Language Basics – Variables

More examples:

byte flags = 63;
shortubb = 10133;
int heiner = 234103234;
long dung = -83628735682345;
float fff = 5464.00345f;
float ggg = -345545.34534E-12f;
double sss = 3245343455.55567d;

char ccc = 'm';
char ccc2 = '\n';  // \n means "new line"
boolean isCool = true;

byte typically used for bit-patterns

Reference Type Variables

- Reference type variables "point" to an object of the reference type

```
byte b1 = new Bicycle();
byte b2 = new Bicycle();

b1.cadence = 1150;
b1.speed = 1151;
b1.gear = 1152;
... ...

b2.cadence = 1405;
b2.speed = 1406;
b2.gear = 1407;
... ...
```

boolean c;
c = b1.equals(b2);
   // c == true
   c = (b1 == b2);
   // c == false