Strings

- Initialisation
  - String str4 = "Hello there";
  - String str5 = new String("Goodbye");
  - char data[] = {'a','b','c'};
  - String str6 = new String(data);
- a single character
  - String str7 = "a";

- Length
  - String str = "Hello";
  - len = str.length();

- Operator =
  - String string_one = "Hello";
  - String string_two;
  - string_two = string_one;

- Operator +
  - String str1 = "Hello ";
  - String str2 = "there";
  - String str3 = str1 + str2; // "Hello there"

s10

- Concatenation
  - String str11 = "abcdefghi";
  - String str12 = "0123";
  - String str13 = str11 + str12;
  - “to”.concat(“ge”).concat(“ther”);
  - //together

s10

- Substring
  - String str12 = "abcdefghi";
  - str12.substring(5); //”fghi”
  - str12.substring(5,7); //”fg”

s10

- Replace(string, string)
  - String cadena1 = "En un lugar de La Mancha";
  - String cadena2 = null;
  - cadena2 = cadena1.replace("lugar", "bar");
  - System.out.println(cadena2);
  - // En un bar de La Mancha

s10

- ReplaceAll (regexp, string)
  - String str11 = "abcdefghi";
  - str11.replaceAll("[abc|g-i]", "1")
  - // 111def111

- Comparison
  - The comparison operators return a **bool** (true/false) value indicating whether the specified relationship exists between the two operands. The operands may be:
    - two string objects

s10

- Equals
  - String string1 = "foo";
  - String string2 = "FOO";
  - if(string1.equals(string2)) {}
  - //false

- EqualsIgnoreCase
  - String string1 = "foo";
  - String string2 = "FOO";
  - if(string1.equalsIgnoreCase(string2)) {}
  - //true

- CompareTo
  - String string1 = "foo";
  - String string2 = "FOO";
  - if(string1.compareTo(string2) == 0) {}
  - //true

- String string1 = "Hello";
- String string2 = "Hello";
- Does the following return true or false?
  - if(string1 == string2) {}

- charAt(index)
  - Returns the character at the given index
  - String string1 = "Hello";
  - string1.charAt(1);
  - // e

- Activity
  - Write a program that receives as input a telphone number as a string. It filters ' ' and '-'. And validates that it is a ten-digit long number and that all characters are digits. If the validation is successful, it prints "valid number"; otherwise it prints "invalid number".

Classes and Objects

s17

Abstract Data Type

- Definition
  - A set of data values and associated operations that are precisely specified independent of any particular implementation
- Examples:
  - Stack, queue, dictionary, etc

s17

Abstract Data Type

- When realized in a computer program
  - the ADT is represented by an interface
  - The impl. shields a corresponding implementation.
- Users of an ADT
  - are concerned with the interface
  - not the implementation, as the implementation can change in the future.
  - This supports the principle of information hiding, or protecting the program from design decisions that are subject to change.

s17

Abstract Data Type

- There's a difference between a data structure and a ADT
  - A data structure is only able to define a set of data
  - An ADT has the possibility of including functions as members, instead of only data

s17

Abstract Data Type

- Example: Stack
  - Operations to modify the data type
    - new() returns a stack
    - popOff(push(v, S)) = S
    - top(push(v, S)) = v
  - Observer operations
    - isEmpty(new()) = true
    - isEmpty(push(v, S)) = false

s17

Classes

- A class is used to define abstract data types
- A class is a set of data and a set of functions able to manipulate the data
- Members of a Class
  - Data
  - Methods

s17

Classes

class *class_name* **{**
// data
class variables
instancace variables

//methods

}

s17

Access modifiers

- Class
  - Public
  - Default
- Class Members
  - Public
  - Protected
  - Private
  - Default
- Local variables
  - None

s17

- If no permission label included in a class, it is considered default
- If no permission label included, members are considered default

s17

- Some Variables may be:
  - Public, static and final
- Example:
  - Math.PI, Math.E
  - These variables are declared as public, static and final

Cosntructor

- Use same name as the class name
- Used to initilise instance variables and class variables
- If not defined, the compiler creates one without arguments
- If one defined, the compiler will not create one

- The constructor of a class always calls the constructor of its superclass
  - By invoking super()

Hose h = new Horse();
What really happens when you say new Horse()?
Assume Horse extends Animal and Anial extens Object

4. Object()
3. Animal() calls super()
2. Horse() calls super()
1. main() calls new horse()

- Does the following code compile?

```
class A {
    A(int x) {}
}

class B extends A {
    B() {}
}
```

- To solve this you can either
  - Define a constructor in A without aguments
  - Define a constructur in B and invoke super() including the arguments of one of the constructors defined in A

```
public class A {
   A(int x) {}
   A() {}
   public static void main(String args[]) {
       B b = new B();
   }

}

class B extends A {
   B(){ }
}
```

```
public class A {
   A(int x) {}
   public static void main(String args[]) {
       B b = new B();
   }

}

class B extends A {
   B(){ super(5); }
}
```

Constructor

- Invoked with "new" to create instances of classes
  - Allocates memory to access the object
- There can be several constructors, distinguised by different arguments
- Can be public, private, protected and default
- Do not have return type

Constructor

- No **Destructor** is used in Java
  - It is not necessary as there is a garbage collection mechanism

s17

- Example: constructor

```
class TestConstructor {
    int var1;
    double var2;
    String var3;

    public TestConstructor() { var3 = "Hello"; }

    public void test() {
        System.out.println("var1= " + var1);
        System.out.println("var2= " + var2);
        System.out.println("var3= " + var3);
    }

}
```

s17

```
public class Constructor {
    public static void  main(String args[]) {
        TestConstructor testConstructor1 = new TestConstructor();
        TestConstructor testConstructor2 = new TestConstructor();
        TestConstructor testConstructor3 = new TestConstructor();

        testConstructor1.test();
        testConstructor2.test();
        testConstructor3.test();
    }
}
```

Packages

- Mechanism to organise classes into namespaces
- Classes can be organised into a category providing similar functionality
- Classes in the same package can access each other

Packages

- The package keyword is used to define the package of the classes contained in a file .java
  - package java.awt.event;
- The import keyword is used to use a class contained in another package
  - import java.awt.event.*;
  - import java.awt.event.ActionEvent;

Packages

- Usually defined using a hierarchical naming pattern
  - Separated by a "."
  - Lower levels are called subpackages
  - A subdirectory needs to be created for each subpackage name: java.awt.event
    - java
      - awt
        - event

Packages

- Which one copiles faster?

```
import java.util.*;
//
...
HashMap map;
...
```

```
import java.util.HashMap;
//
...
HashMap map;
...
```

Packages

- It takes the same time to compile
- The import statement only tells the compiler where to look up if a class is not found
- To improve readablility it is prefered:
  - import java.util.HashMap;

Packages

- Activity:Use the command promt to compile and do not use an IDE to edit the files. Define "public class Operations{}" including the method "public int add(int x, int y) {}" as part of the package MyUtilities.math. Implement the method "add" and define the following class to test it:

```
import MyUtilities.math.Operations;

public class TestPackage {
   public static void main(String args[]) {
     Operations operations = new Operations();
      System.out.println("result = " + operations.add(3,5));
   }
}
```