ФАКУЛТЕТ ЗА ИНФОРМАТИЧКИ НАУКИ И КОМПЈУТЕРСКО ИНЖЕНЕРСТВО

Introduction to Java

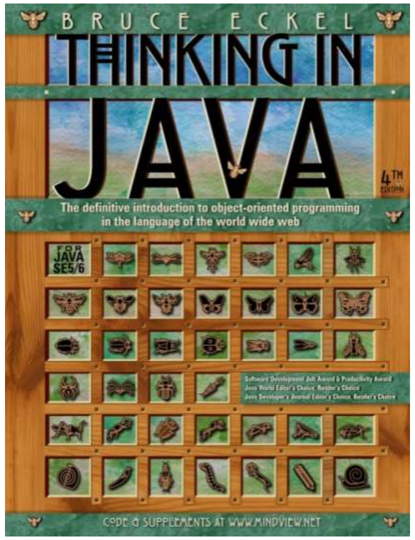
Algorithms and data structures

Exercise 0

Why Java?

- Developed by Sun Microsystems, 1995 (James Gosling)
- Object-oriented language for general purposes
- Based on C/C++
- Designed for easy Web/Internet applications
- Widely used
- Platform/Operating system independent

Literature


BRUCE ECKEL
THINKING IN JAVA
4TH EDITION
The definitive introduction to object-oriented programming in the language of the world wide web
FOR JAVA SE5/6
CODE & SUPPLEMENTS AT WWW.MINDVIEW.NET

- Thinking in Java, 4th edition
  – by Bruce Eckel
  – http://mindview.net/Books/TIJ4

Web resources…

- http://www.apl.jhu.edu/~hall/java/

Java characteristics

- **Simple**
  - Solves part of the problems from C++
  - No pointers
  - Automatic garbage collection
  - Rich predefined class libraries http://java.sun.com/j2se/1.4.2/docs/api/
- **Object oriented**
  - Focus on data (objects) and methods (functions) which manipulate with data
  - Each function is with an object
  - All data types are objects (files, strings etc.)
  - Good code organization and reusability

Java disadvantages

- **Slower than compiling languages as C**
  - Experiments show that Java is 3 or 4 times slower than C or C++

    read: "Comparing Java vs. C/C++ Efficiency Issues to Interpersonal Issues" (Lutz Prechelt)
  - Good for many apps, except for time critical ones

Development environment

- **There are many programs that give support for Java software development, including:**
  - Sun Java Development Kit (JDK)
  - Sun NetBeans
  - IBM Eclipse, GNU Eclipse
  - Borland JBuilder
  - MetroWerks CodeWarrior
  - BlueJ
  - jGRASP

}Installed in FINKI labs

- **Althought there are environmental design details differences, the basic compiling and execution process is identical**

Eclipse

- Eclipse is a graphical IDE developed by Eclipse Fondation
- The code for Eclipse is written in Java
- Eclipse is developed as a platform not only for Java programming language, but for so called plug-ins (one of them is Java itself)

Easy installation

- Download from www.eclipse.org
- Recommend to download Eclipse IDE for Java EE Developers
- Newest version Eclipse
- Attention!
- Before installing Eclipse install JDK (Java Development Kit)

Easy installation

- No need for installation
- Only unzip and start
- At first Eclipse asks for the location of your workspace directory where you have all of your projects and files

Work with Eclipse

- How to write and execute Java code in Eclipse in 5 steps:
  – Make new project (Java Project)
  – Add class to the project (File->New->Class)
  – Write the code
  – Compile is automatic, no need you doing that
  – Run the project(Run->Run…)
  – Debug – find your errors (Run->Debug…)

First program in Java

```
public class Hello
   {
      public static void main(String[] args)
      {
         System.out.println("Hello world!\n");
      }  // end of main
   }  // end of class Hello
```

- public static void main(String[] args) is a part of every Java program

- The program starts with execution in main
- Main is a function
- Java apps have one or more functions
- Only one function is called main

Attention:

JAVA is case-sensitive

System != system != SyStEM

Important for Java code

- In a Java program there has to be one public class
- **public static void main (String [ ] a)** is mandatory in the public class, where you write the code

Ex.
```
public class test {
    public static void main (String [] a)
   {
                    write the code
    } //end of main
   } //end of class
```

Everything is an object

- In Java, everything is an object, even programs
- C++ is a hybrid language
  - C compatible
- In Java, the only way of programming is object-oriented programming
- An object is created with “new” keyword

```
String s = new String(“Hello!”);
```

Data types

- Basic (primitive) vs. reference data types
- Basic, primitive types
  - boolean, byte, char, short, int, long, float, double
  - All have default values
- Non primitive data types in Java are objects and arrays.
  - They are called reference types (by reference)
  - Default value of each reference is null

Object destroy

- No need!!!!
- Garbage Collection is automatic
- Java programmer is free of duty to control memory

Rules

- Every rule for object-oriented languages, is in Java
- Course from OOP!!!
- Rules are the same, only the syntax in the language is different.

Packages in Java

- Java Application Programming Interface (API)
  - Java Class Library
  - Contains predefined methods and classes
    - Similar classes are organized in packages
    - Includes mathematical computation methods, strings/arrays manipulation methods, input/output, databases, network working, files processing, error correction methods, etc.
- Class and methods collection from Java API (http://docs.oracle.com/javase/8/docs/api/)

Input/output in Java

- Output command:

```
System.out.println("Java programming is interesting.");
```

- Printing variables (objects):

```
Integer number = 10;
System.out.println("Number = " + number);
Дјдфх
```

```
String s = new String("I'm");
System.out.println(s + " FINKI student");Дјдфх
```

Input/output in Java

- Scanner class use

```
 import java.util.Scanner;
```

- A Scanner class object is created and it is used to get user input

```
Scanner input = new Scanner(System.in);
int number = input.nextInt();
Дјдфх
```

- After read ending close() method is called to close the object

Input/output in Java

- Example: Read integer input from user

```
import java.util.Scanner;
 class Input {
      public static void main(String[] args) {
             Scanner input = new Scanner(System.in);

             System.out.print("Enter an integer: ");
             int number = input.nextInt();
             System.out.println("You entered " + number);

             // closing the scanner object
             input.close();
      }
}
```

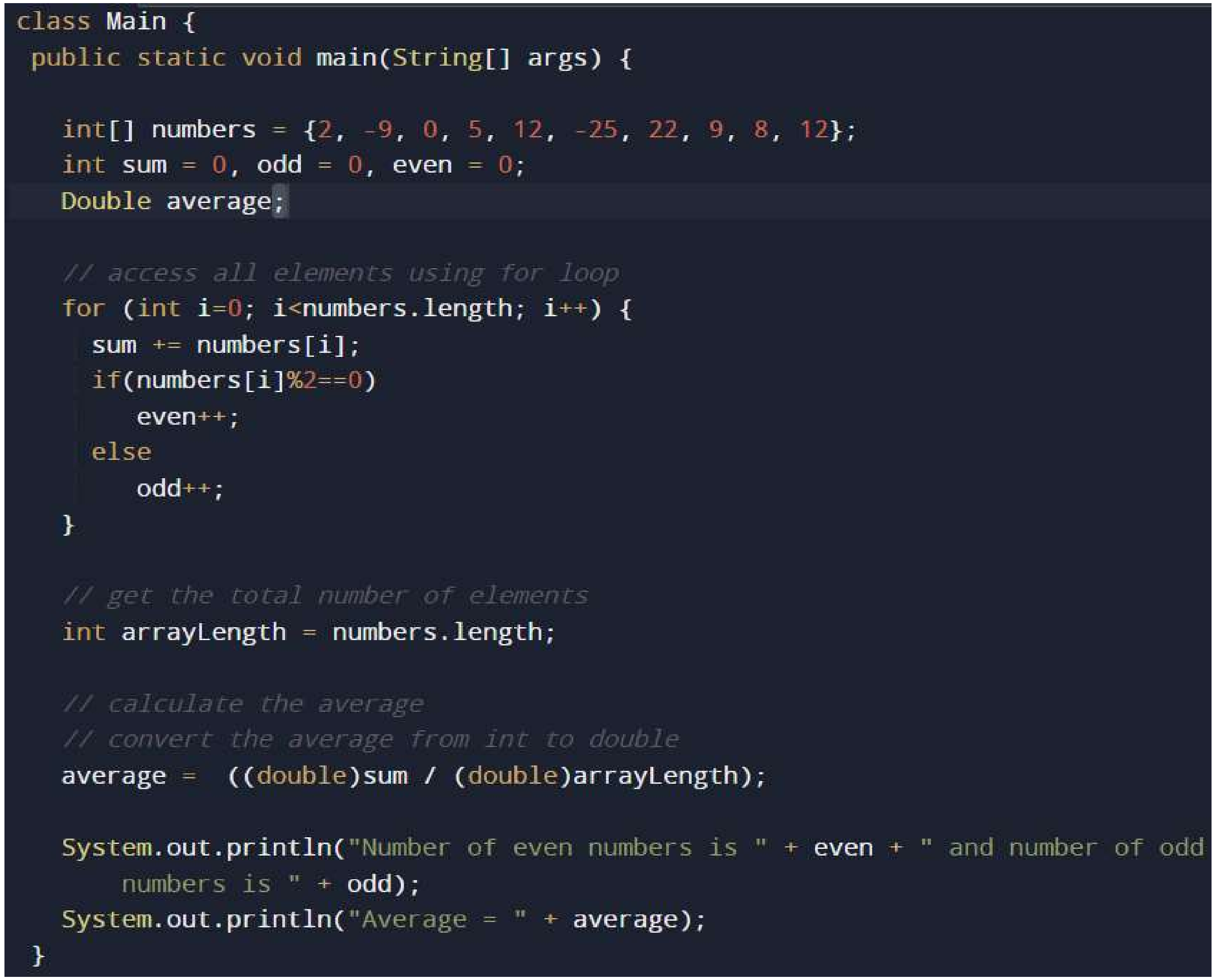
Input/output in Java

- Example: Read input given in one line (until enter input)

```
import java.util.Scanner;
 class Input {
      public static void main(String[] args) {
             Scanner input = new Scanner(System.in);
             System.out.print("Enter your name: ");

             // reads the entire line
             String value = input.nextLine();
             System.out.println("Using nextLine(): " + value);

             // closing the scanner object
             input.close();
      }
}
```

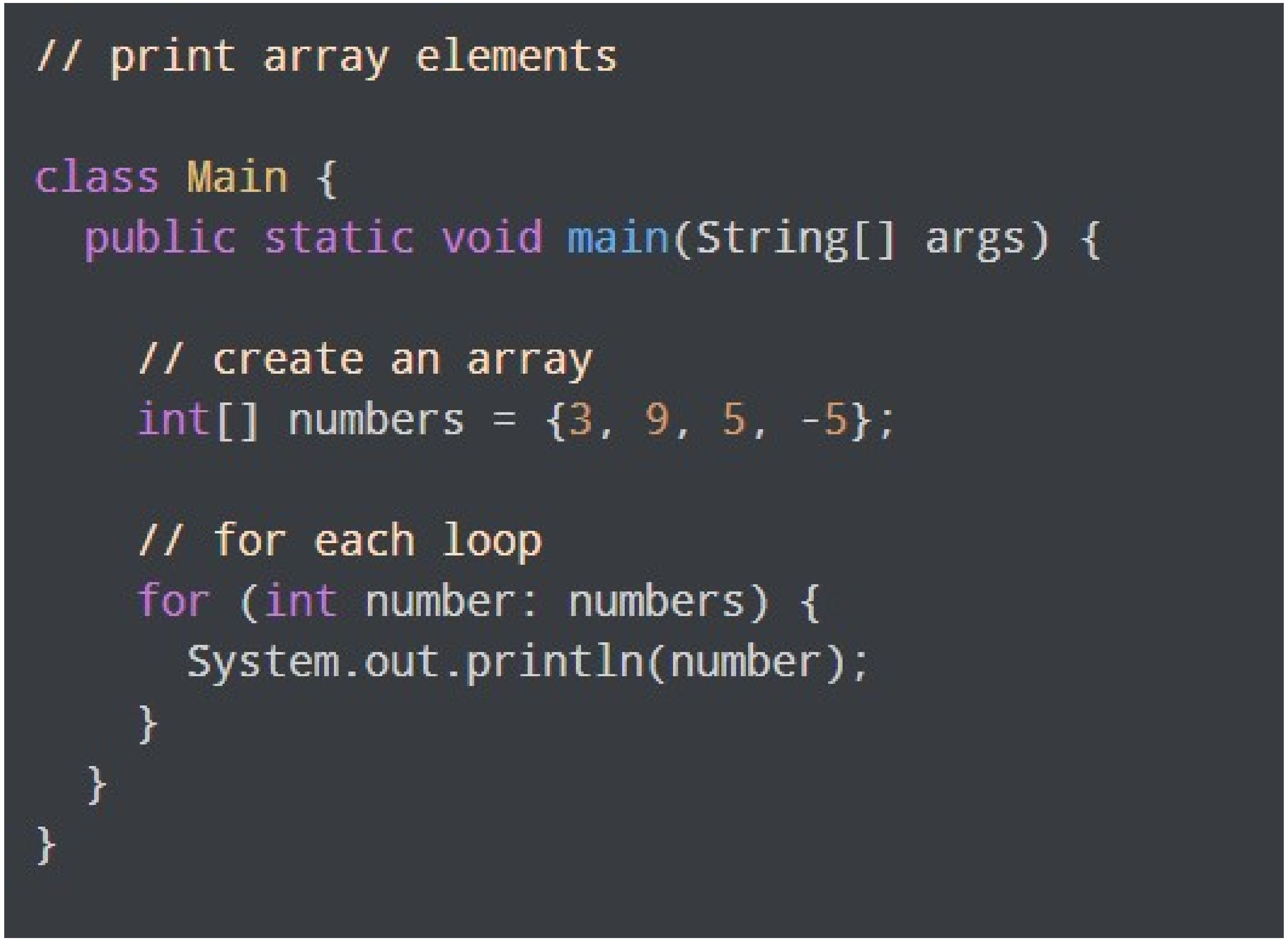
Control flow in Java

Example 1: Print the number of even and odd numbers as well as their average from a given input integers.



```
class Main {
 public static void main(String[] args) {

   int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
   int sum = 0, odd = 0, even = 0;
   Double average;

   // access all elements using for loop
   for (int i=0; i<numbers.length; i++) {
     sum += numbers[i];
     if(numbers[i]%2==0)
        even++;
     else
        odd++;
   }

   // get the total number of elements
   int arrayLength = numbers.length;

   // calculate the average
   // convert the average from int to double
   average =  ((double)sum / (double)arrayLength);

   System.out.println("Number of even numbers is " + even + " and number of odd
       numbers is " + odd);
   System.out.println("Average = " + average);
 }
```

For-each structure



```java
// print array elements

class Main {
  public static void main(String[] args) {

    // create an array
    int[] numbers = {3, 9, 5, -5};

    // for each loop
    for (int number: numbers) {
      System.out.println(number);
    }
  }
}
```

Repeat structures

for *cycle:*

```
for( <initialization> ; <condition> ; <statement> ){

        <Block of statements>;

}
```

while *cycle:*

```
while(<boolean condition>){

        <Block of statements>;

}
```

do-while *cycle:*

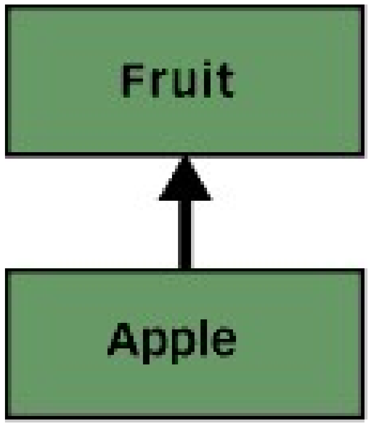
```
do{

        <Block of statements>;

}while(<boolean condition>);
```

Inheritance

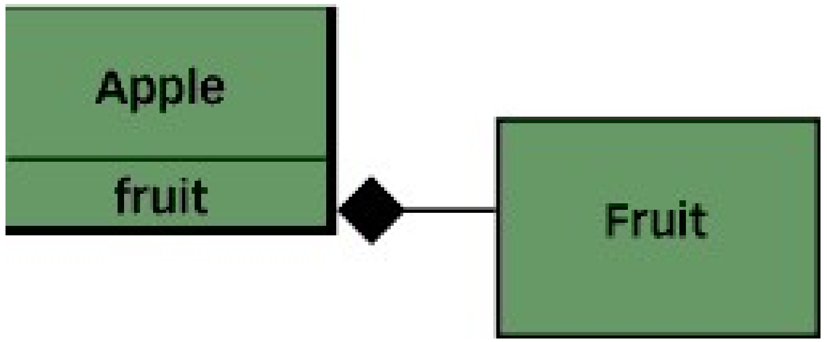
- Class B inherits class A if objects from class B have the same properties as class A, with new characteristics
- In Java the subclass expands the superclass.
- Inheritance: “is”-relation

“is-a” vs. “has-a”

- Don’t mix is-a with has-a!
- has-a means that one class includes objects from another class as attributes



Inheritance



Composition

Differences in Java

- **Terminology**
  - **Superclass** - A (basic, parental class)
  - **Subclass** – B (derived class)

- In Java
  - Only one inheritance - a hierarchy tree
  - No multi-inheritance
    - When a class inherits from different classes
    - Solved with Interface

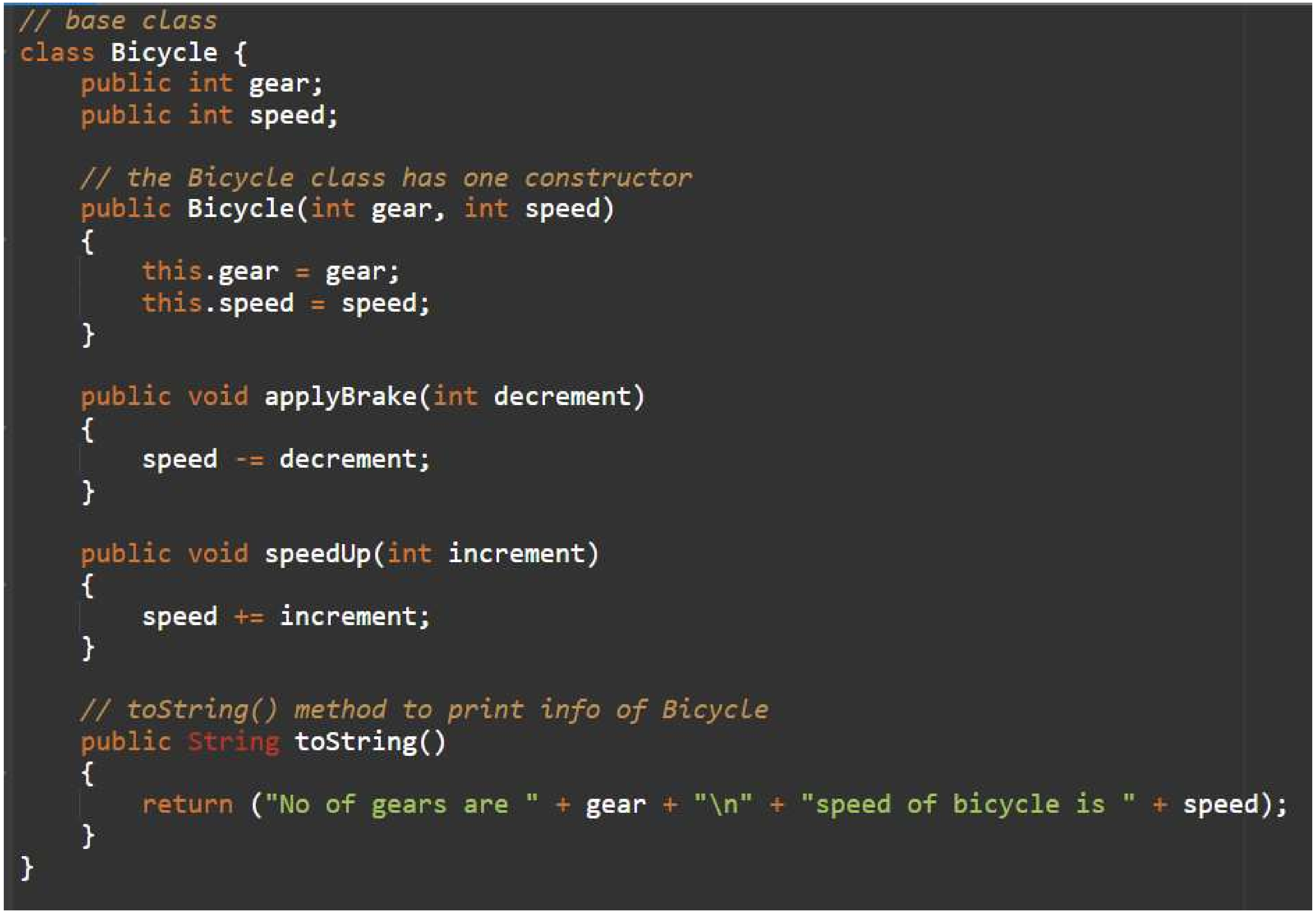
Class hierarchy

- Subclass inherits from superclass, but adds its own properties
  - Adds its own attributes
  - Adds its own methods
- Direct superclass
  - Subclass is below superclass
- Indirect superclass
  - Is every superclass which is not directly connected with the subclass in a class hierarchy

Basic class in Java

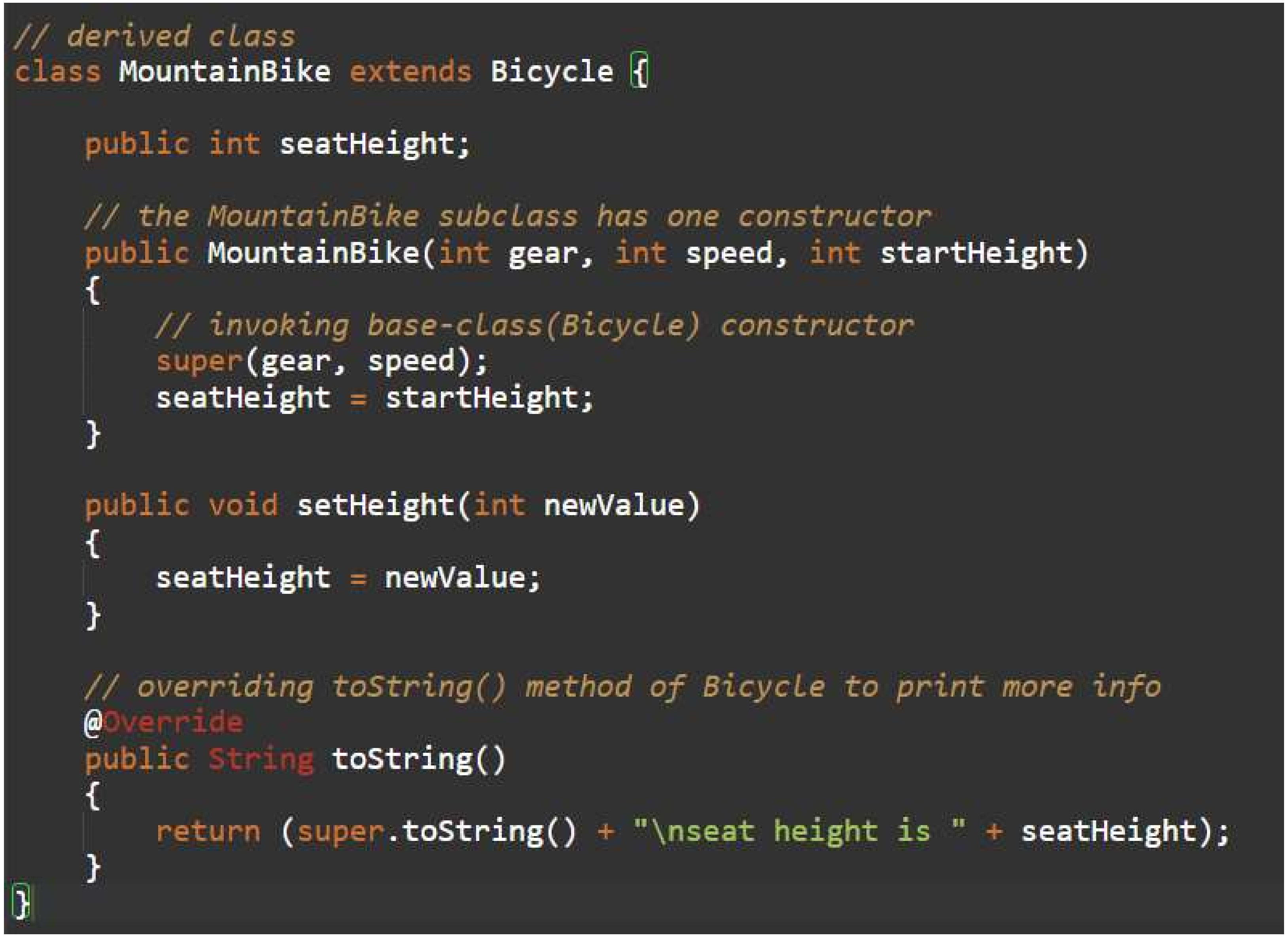
- Class hierarchy adds class relations
- Class hierarchy starts with the class **Object** (in package java.lang)
  - .. From which each class in Java is inherited (directly or indirectly)
- For a superclass of a given class Java compiler sets the class Object in case when class declaration doesn’t give an explicit definition from another class

Class hierarchy - example



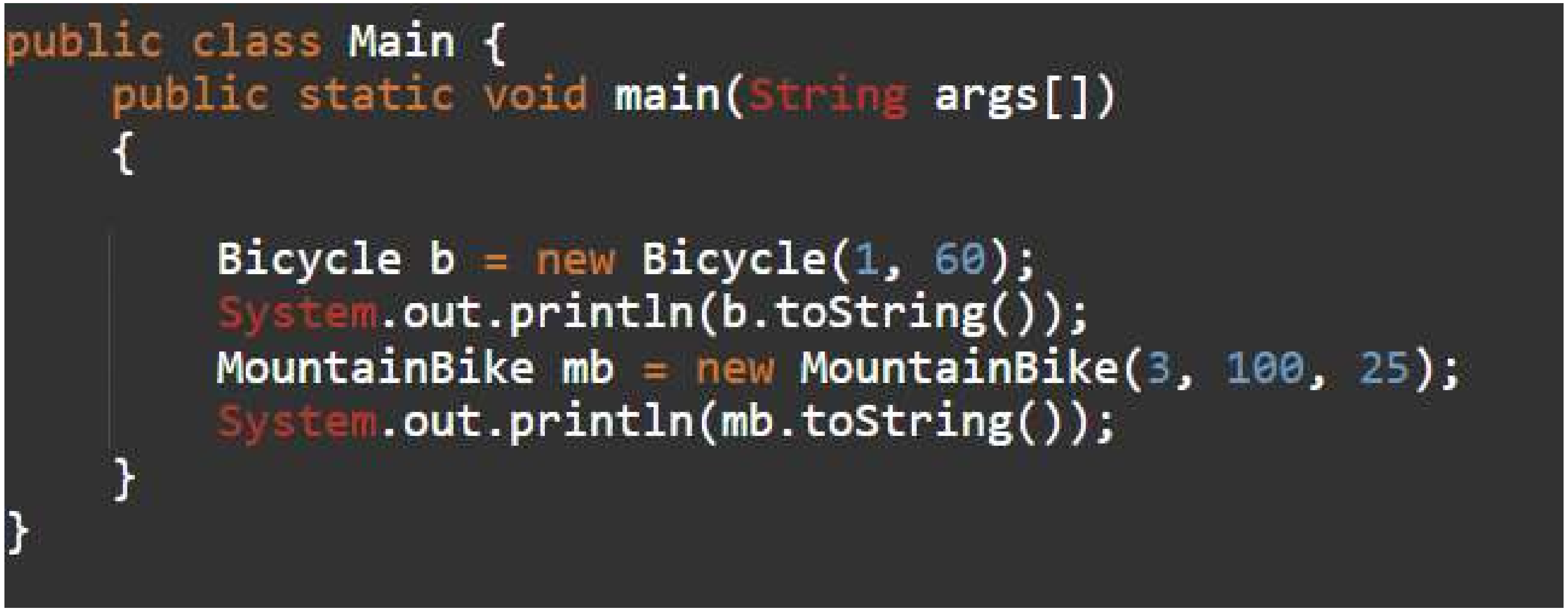
```java
// base class
class Bicycle {
    public int gear;
    public int speed;

    // the Bicycle class has one constructor
    public Bicycle(int gear, int speed)
    {
        this.gear = gear;
        this.speed = speed;
    }

    public void applyBrake(int decrement)
    {
        speed -= decrement;
    }

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

    // toString() method to print info of Bicycle
    public String toString()
    {
        return ("No of gears are " + gear + "\n" + "speed of bicycle is " + speed);
    }
}
```

Class hierarchy - example



```java
// derived class
class MountainBike extends Bicycle {

    public int seatHeight;

    // the MountainBike subclass has one constructor
    public MountainBike(int gear, int speed, int startHeight)
    {
        // invoking base-class(Bicycle) constructor
        super(gear, speed);
        seatHeight = startHeight;
    }

    public void setHeight(int newValue)
    {
        seatHeight = newValue;
    }

    // overriding toString() method of Bicycle to print more info
    @Override
    public String toString()
    {
        return (super.toString() + "\nseat height is " + seatHeight);
    }
}
```

Class hierarchy - example



```
public class Main {
    public static void main(String args[])
    {

        Bicycle b = new Bicycle(1, 60);
        System.out.println(b.toString());
        MountainBike mb = new MountainBike(3, 100, 25);
        System.out.println(mb.toString());
    }
}
```

What is Generics?

- In a nutshell, generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods.
- Much like the more familiar *formal parameters* used in method declarations, type parameters provide a way for you to re-use the same code with different inputs

Generic Types

- A *generic type* is a generic class or interface that is parameterized over types
- For example:
  - LinkedList<E> has a type parameter E that represents the type of the elements stored in the linked list

Why Generics?

- Stronger type checks at compile time
  - Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find
- Elimination of casts
- Enabling programmers to implement generic algorithms
  - By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read

Example 1

- Code without Generics requires casting

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0); //casting
```

- Code rewritten to use Generics

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

Example 2

- Code without generics is not type safe code

```
Vector v = new Vector();
v.add(new String("hello"));
v.add(new Integer(5));
// ClassCastException occurs during runtime
String s = (String) v.get(1);
```

- With generics, it is safe

```
Vector<String> vs = new Vector<String>();
vs.add(new Integer(5));  // compile error!
vs.add(new String("hello"));
String s = vs.get(0);
```

Naming Convention

- By convention, type parameter names are single, uppercase letters
- The most commonly used type parameter names are:
  - E - element
  - K - key
  - N - number
  - T - type
  - V - value
  - S, U, V – 2nd, 3rd, 4th types

Example

- A simple `Box` class

```
public class Box {
   private Object object;

   public void set(Object object){this.object = object;}
   public Object get() { return object; }
}
```

- A generic version of the `Box` class

```
public class Box<T> {
  // T stands for "Type"
  private T t;

  public void set(T t) { this.t = t; }
  public T get() { return t; }
}
```

Instantiating a Generic Type

- A simple `Box` class

```
public class Box {
   private Object object;

   public void set(Object object){this.object = object;}
   public Object get() { return object; }
}
```

```
Box b = new Box();
```

- A generic version of the `Box` class

```
public class Box<T> {
  // T stands for "Type"
  private T t;

  public void set(T t) { this.t = t; }
  public T get() { return t; }
}
```

```
Box<Integer> intBox = new Box<Integer>();
```

Multiple Type Parameters

- A generic class can have multiple type parameters
- For example, the generic `OrderedPair` class, which implements the generic `Pair` interface:

```
public interface Pair<K, V> {
     public K getKey();
     public V getValue();
}
```

Multiple Type Parameters

```
public class OrderedPair<K, V> implements
Pair<K, V> {
   private K key;
   private V value;

   public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
   }

   public K getKey(){ return key; }
   public V getValue() { return value; }
}
```

- Two instances of the class `OrderedPair`

```
OrderedPair<String, Integer> p1 = new OrderedPair<>("Even", 8);
OrderedPair<String, String>  p2 = new OrderedPair<>("hello", "world");
```

Generics and Sub-typing

- You can do this :

```
Number someNumber = new Number();
Integer someInteger = new Integer(10);
someNumber = someInteger;    // OK
```

  – According to the OOP principles, `Integer` is a direct subclass of `Number`

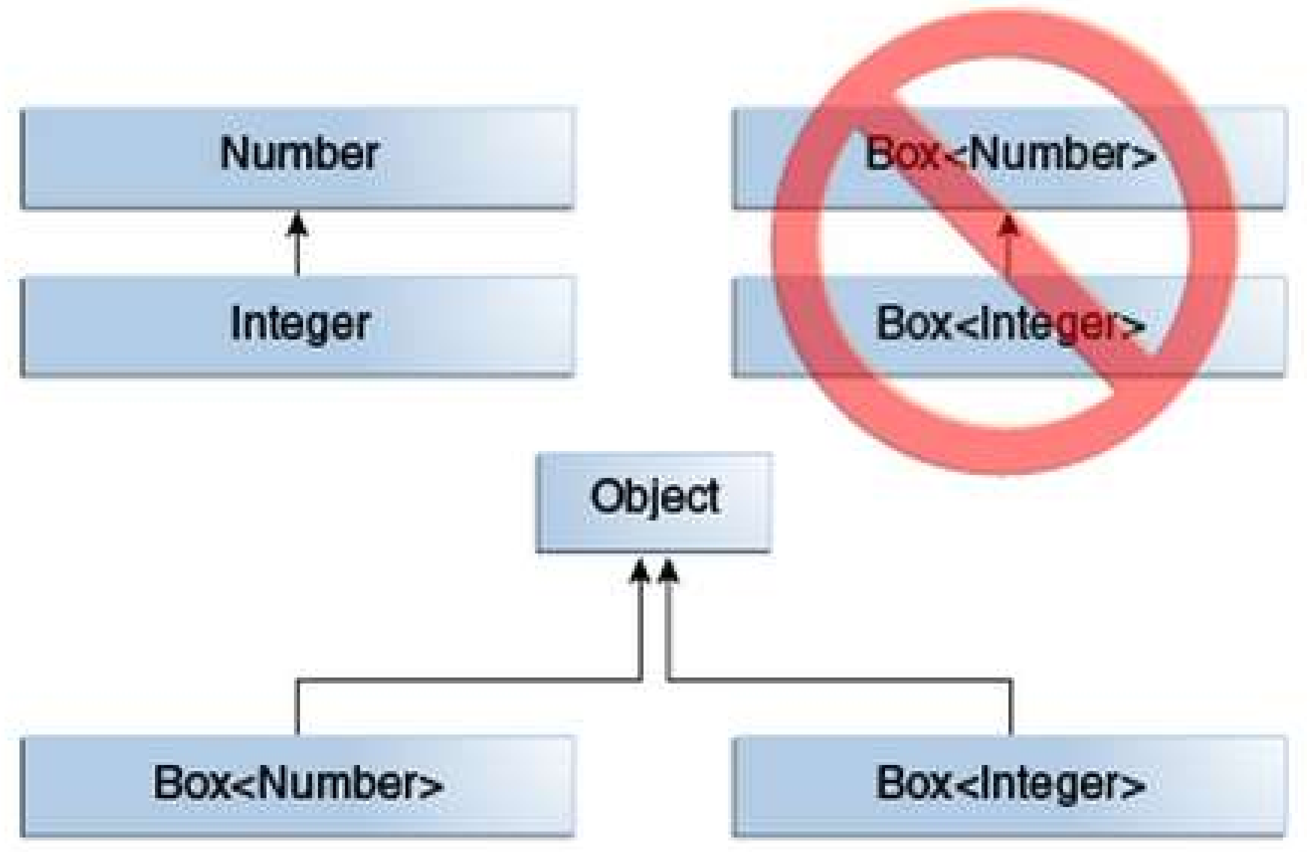
- So you would expect to be able to do this

```
Box<Number> box = new Box<Integer>();
```

  – Well, you can't do this!!!

Generics and Sub-typing

- So there is **no inheritance relationship between type arguments** of a generic class



Generics and Sub-typing

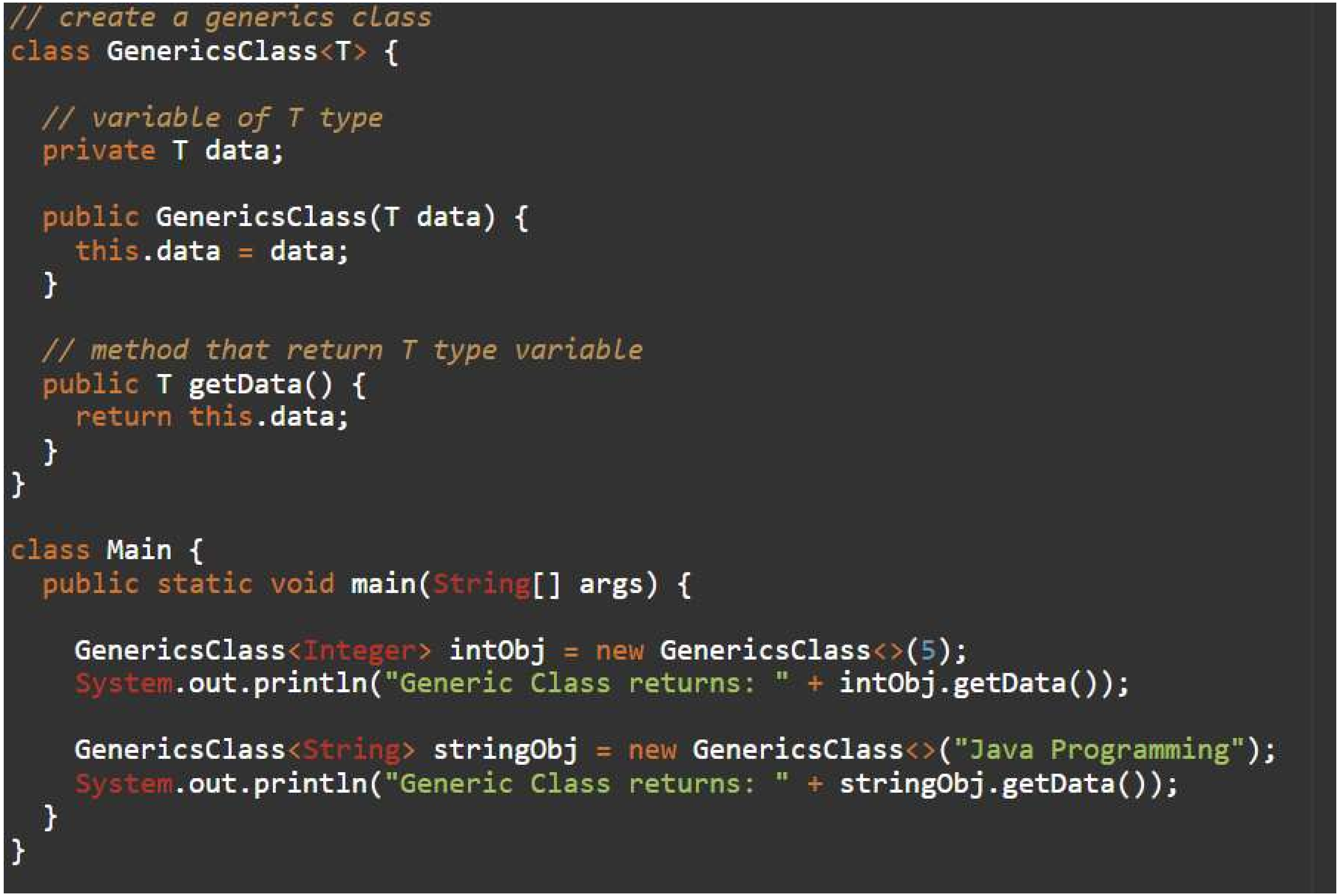
- Entries in a collection maintain inheritance relationship
- The following code is valid

```
ArrayList<Number> an = new ArrayList<Number>();
an.add(new Integer(5)); // OK
an.add(new Long(1000L)); // OK
```

  – But, this is not!

```
an.add(new String(“hello”)); // compile error
```

Generics example



```java
// create a generics class
class GenericsClass<T> {

  // variable of T type
  private T data;

  public GenericsClass(T data) {
    this.data = data;
  }

  // method that return T type variable
  public T getData() {
    return this.data;
  }
}

class Main {
  public static void main(String[] args) {

    GenericsClass<Integer> intObj = new GenericsClass<>(5);
    System.out.println("Generic Class returns: " + intObj.getData());

    GenericsClass<String> stringObj = new GenericsClass<>("Java Programming");
    System.out.println("Generic Class returns: " + stringObj.getData());
  }
}
```