Part II

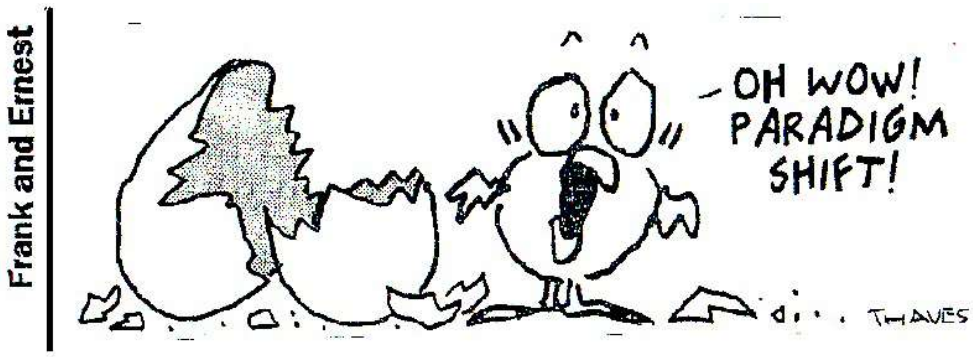
Differences between Java and C++

James Gosling

Bjarne Stroustrup

Programming Paradigms

- C++ was developed as an extension of C and it is a hybrid of two distinct programming paradigms: procedural and object-oriented paradigms
  - In C++, it is possible to have variables and functions, which are declared outside of any class
- Different from C++, Java was designed from ground up as an object-oriented language, not a hybrid
  - In Java, everything in a program is part of some class. Because of this, Java has neither global data (i.e. variables and objects) nor global functions



Programming Paradigms - C++

```cpp
// Filename: Fib.cpp
#include <iostream>
using namespace std;

int fibCallsCounter; // Global variable

int fib(int n) {     // Global function
  fibCallsCounter++;
  return (n <= 2) ? 1 : fib(n-1) + fib(n-2);
}

int main(int argc, char* argv[]) {
  while(true) {
    int n;
    fibCallsCounter = 0; // Refer to global variable
    cout << "Desired value of n (0 to quite): ";
    cin >> n;
    if(n == 0)
      break;
    // fib(n) refers to a global function call
    cout << "The " << n << "th fibonacci number is " << fib(n) << endl;
    cout << "Function was called " << fibCallsCounter << " times" << endl;
  }
}
```

Programming Paradigms - Java

```java
// Filename: Fib.java
import java.util.Scanner;
import java.io.IOException;

public class Fib {
  private static int fibCallsCounter;
  private static int fib(int n) {
    fibCallsCounter++;
    return (n <= 2) ? 1 : fib(n-1) + fib(n-2);
  }
  public static void main(String[] args) throws IOException {
    Scanner sc = new Scanner(System.in);
    while(true) {
      int n;
      fibCallsCounter = 0;
      System.out.print("Desired value of n (0 to quite): ");
       n = sc.nextInt();
      if(n == 0)
        break;
      System.out.println("The " + n + "th fibonacci number is " + fib(n));
      System.out.println("Function was called " + fibCallsCounter + " times");
    }
    sc.close();
  }
}
```

Primitive Data Types

- C++ and Java provide similar data types but they do not use exactly the same type names
- The table below shows all the primitive data types supported by Java

Type	Size	Range
byte	1 byte	-128 to 127
short	2 bytes	$-32{,}768$ to 32,767
int	4 bytes	$-2{,}147{,}483{,}648$ to 2,147,483,647
long	8 bytes	$-9{,}223{,}372{,}036{,}854{,}775{,}808$ to 9,223,372,036,854,775,807
float	4 bytes	$-$ve range: $-3.4028235 \times 10^{38}$ to -1.4×10^{-45} $+$ve range: 1.4×10^{-45} to 3.4028235×10^{38}
double	8 bytes	$-$ve range: $-1.7976931348623157 \times 10^{308}$ to -4.9×10^{-324} $+$ve range: 4.9×10^{-324} to $1.7976931348623157 \times 10^{308}$
char	2 bytes	0 to 65,535 (unsigned)
boolean	not precisely defined	true or false

Types other than the 8 primitive data types are non-primitive

Primitive Data Types

More details about Java console I/O will be discussed

- C++ regards the type char as an integer type. Thus char can be used as a one-byte integer in some contexts, which is equivalent to the Java type byte

C++

```
char val = 10;
cout << "val: " << (int)val << endl;
```

Java

```
byte val = 10;
System.out.println("val: " + val);
```

- Type-checking and type requirements are much tigher in Java. For example:
  - Conditional expressions (e.g. those in if, while and do-while) can be only boolean, not integral

Java - Wrong (with compilation error)

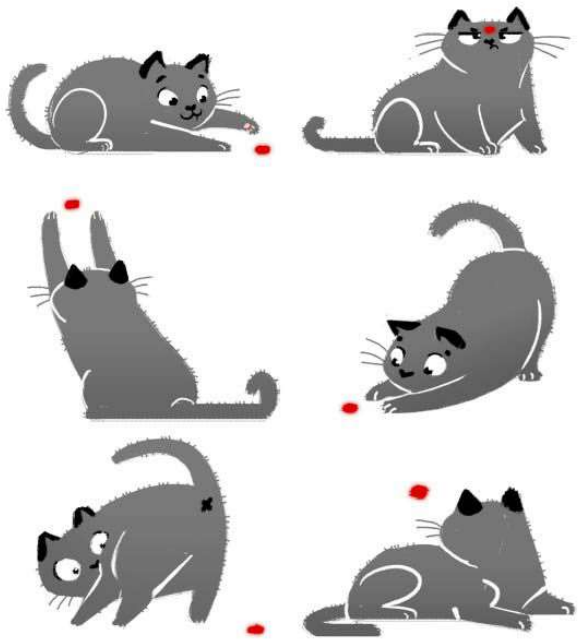
```
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();
if(x)
  System.out.println("true");
else
  System.out.println("false");
```

Java - Correct

```
Scanner sc = new Scanner(System.in);
int x = sc.nextInt();
if(x != 0)
  System.out.println("true");
else
  System.out.println("false");
```

Non-Primitive Data Types - Reference Types

- Types other than the 8 primitive data types are non-primitive
- Non-primitive types are called reference types
- Reference types in Java are something similar to C++ pointers, but
  - No asterisk operator is needed when declaring reference
  - No arithmetic can be done with references
  - No dereference can be done with references



Reference Types: Question and Answers

- Is Scanner a primitive type?
  - No. It is a non-primitive type (or reference type)
- How to declare a Scanner reference?
  - ```
    Scanner sc; // Remember: No asterisk is needed!
        // Also, no object of Scanner type is instantiated
    ```
- How can it "point" at a Scanner object?
  - ```
    sc = new Scanner(System.in);
    ```
- Can it "point" at null?
  - Yes, do the following.

    ```
    sc = null; // Remember: All letters should be in lowercase
    ```
- Can we do this?

  ```
  sc++;
  ```

  - No. Arithmetic cannot be done with references

Note

Java references are not related to C++ references at all

Instantiation of Variables or Objects

- C++ allows instantiation of variables or objects of all types (both primitive and non-primitive) using new keyword or without
- But Java requires all variables of primitive types (i.e. byte, short, int, long, float, double, char, boolean) to be instantiated without using new keyword, and all objects (i.e. non-primitive types) to be instantiated using new

C++

```
int x = 10;
int* y = new int;

Person* p = new Person("Tom", 'M', 18);

Person q; // An object
          // is instantiated
```

Java

```
int x = 10;
// Nothing equivalent

Person p = new Person("Tom", 'M', 18);

Person q; // Correct, but no object
          // is instantiated
```

Array Types

- C++ allows instantiation of arrays using new keyword or without
- Java only allows instantiation of arrays using new keyword

C++

```
int arr[10];
int* p = new int[10];
int* q = new int[20];
int* r; // Only defines a pointer
int* s; // Only defines a pointer
Person pArr[30];
Person* pPtr = new Person[30];
```

Java

```
int arr[10]; // Wrong
int p[] = new int[10];
int[] q = new int[20];
int[] r; // Only defines a reference
int s[]; // Only defines a reference
Person pArr[30]; // Wrong
Person[] pPtr = new Person[30];
```

- Bounds checking is performed in Java, but not in C++. When you try to access an element of an array that is out of legal range, runtime error occurs
- In Java, a length member is available in array to tell how many elements are there

Memory Allocation and De-allocation

- In C++, the programmer is responsible for explicitly freeing storage allocated using new by using the corresponding operator delete
- In Java, the programmer is not necessary to free the storage allocated using new. The storage will be reclaimed by Java when there are no more references to the storage.
  - "Garbage Collection"

C++

```
void someFunction() {
  SomeClass* p = new SomeClass;

  // ... Code that uses the object
  //     pointed by p

  // ... Now suppose the object we
  // ... created is no longer needed

  // Free up the space allocated by new
  delete p;
}
```

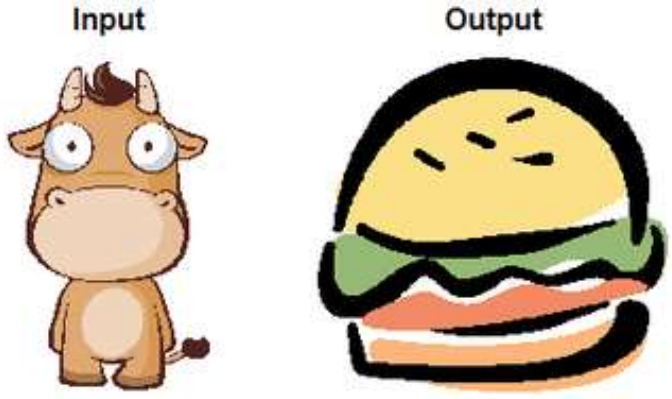
Java

```
void someMethod() {
  SomeClass p = new SomeClass();

  // ... Code that uses the object
  //     referenced by p

  // ... Now suppose the object referenced
  //     by p is no longer needed

  // No need to worry about freeing up
  // space allocated by new
  // Garbage collector will do it for us
}
```

Support for Graphical User Interface (GUI) Input / Output

Details of packages will be discussed later

- C++ standard libraries provide strong support for textual input / output to the standard input and standard output (normally the command line), but they provide no support for GUI input / output
  - To write GUI programs, one must use platform-specific GUI libraries, e.g. X-Windows on Unix platforms, which means GUI programs will require extensive rewriting when being ported from one platform to another
- Java defines a platform-independent set of GUI tools through packages
  - GUI programs written in Java can run without modification on any system supporting Java

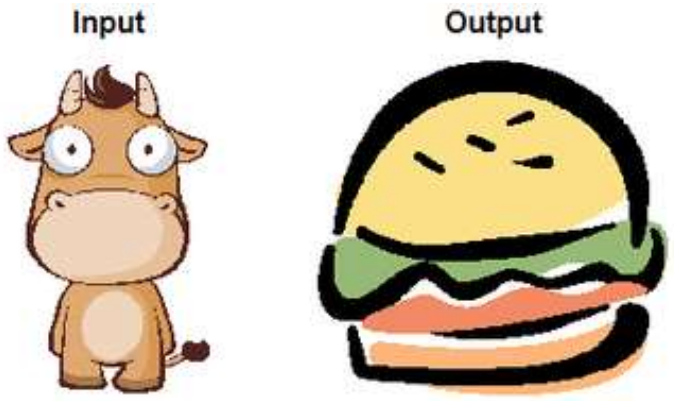


Support for Console Input / Output

- C++ provides operators (<< and >>) for outputting and inputting data, while Java does not support the use of operators for console I/O

```cpp
// C++
int n;
cout << "Enter n: ";
cin >> n;
cout << "Value of n: " << n << endl;
```

```java
// Java
Scanner sc = new Scanner(System.in);
System.out.print("Enter n: ");
int n = sc.nextInt();
System.out.println("Value of n: " + n);
```



Display Data on Monitor's Screen in Java

- Java provides three methods to display data on monitor's screen (specifically is the command window)
  1. System.out.print()
  2. System.out.println()
  3. System.out.printf(...)
     ↑ will not be covered in this course

System.out is a standard output object, i.e., an object with methods that output data to screen

Syntax

```
System.out.print(<data> [ +<data>]);
System.out.println(<data> [ +<data>]);
System.out.println();
```

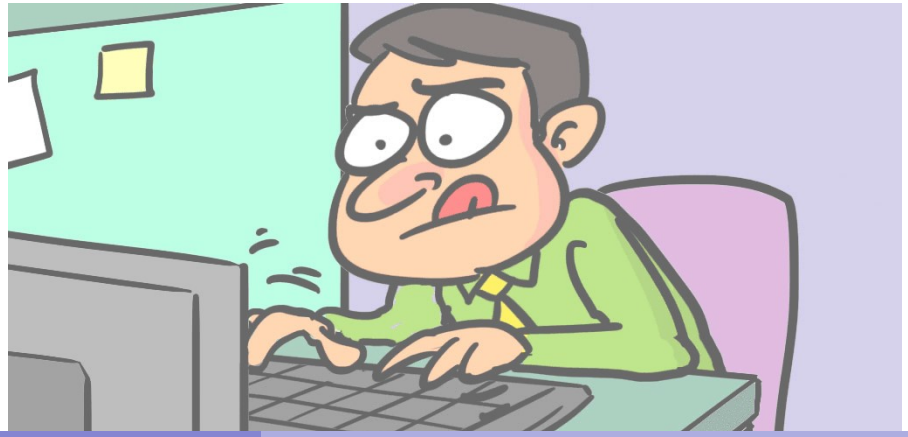
where <data> is the type of data that you would like to display, <data> can be in types: byte, short, int, long, float, double, char, boolean, String, etc. [...] may be repeated zero or more times, but must be separated by +

Display Data on Monitor's Screen in Java

```java
// Filename: OutputDataDemo1.java
/* An example showing how to output data on screen */
public class OutputDataDemo1 {
  public static void main(String[] args) {
    int val = 9;
    System.out.println(val);
    System.out.print("You should be able to see this text ");
    System.out.println("on screen");
    System.out.print("How about this text?");
    System.out.print("Can you see the effect?");
  }
}
```

Screen output:

```
9
You should be able to see this text on screen
How about this text?Can you see the effect?
```

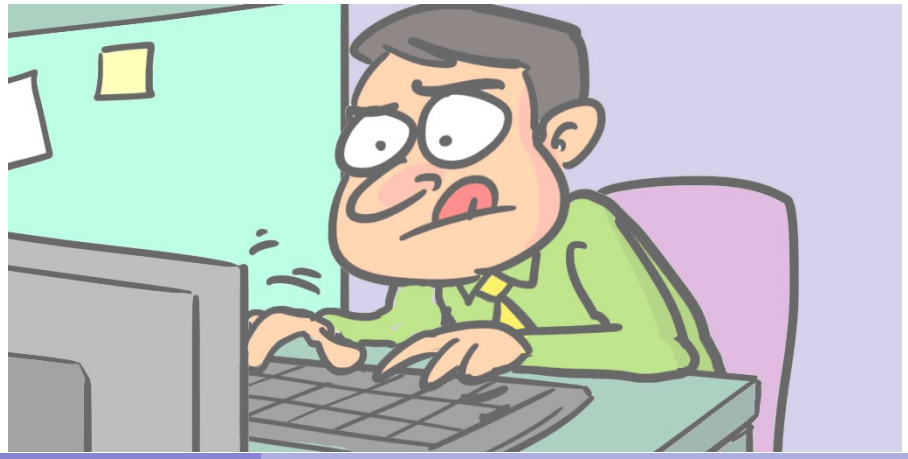


Display Data on Monitor's Screen in Java

```java
// Filename: OutputDataDemo2.java
/* An example showing how to output different types
   of data on screen */
public class OutputDataDemo2 {
  public static void main(String[] args) {
    String str = "Tom";
    char gender = 'M';
    int age = 18;
    double height = 1.71;
    System.out.println("Hi " + str);
    System.out.println("Your age is " + age);
    System.out.println("Your gender is " + gender);
    System.out.println("Your height is " + height + 'm');
  }
}
```

Screen output:

```
Hi Tom
Your age is 18
Your gender is M
Your height is 1.71m
```



Get User's Input from Keyboard in Java

- Java provides a class named "Scanner" for getting user's input from keyboard
- The Scanner class is a part of the package java.util
  - Package is a collection of classes
- To use the Scanner class, you can follow the syntax shown below

Syntax

```
// Put the following line at the top of the Java source file
import java.util.Scanner;

Scanner <object name>  = new Scanner(System.in);
<variable name> = <object name>.<method name>();
<object name>.close();
```

where <object name> is the name of the newly created Scanner class object, new is the keyword to create an object, <variable name> is the name of the variable to store the data obtained from the keyboard using Scanner method

Scanner Class Methods

Method	Description
boolean nextBoolean()	Returns the next input token as a Boolean value
byte nextByte()	Returns the next input token as a byte value
short nextShort()	Returns the next input token as a short value
int nextInt()	Returns the next input token as an int value
long nextLong()	Returns the next input token as a long value
float nextFloat()	Returns the next input token as a float value
double nextDouble()	Returns the next input token as a double value
String next()	Returns the next input token as a String value
String nextLine()	Returns all input remaining on the current line as a String value

- The details in the column "Method" describe two things: name of the method and the type of data returned
- For instance: int nextInt()
  - This means the name of the method for obtaining an int from keyboard is nextInt, and
  - The keyword int means it returns an int, which is the integer it obtains

`https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html`

Example: Get User's Input from Keyboard in Java

```java
// Filename: InputDataDemo.java
/* An example showing how to get input
   from keyboard */
import java.util.Scanner;
public class InputDataDemo {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter your name: ");
    String name = sc.nextLine();
    System.out.print("Enter your age: ");
    int age = sc.nextInt();
    System.out.print("Enter your gender: ");
    String gender = sc.next();
    System.out.print("Enter your height: ");
    double height = sc.nextDouble();
    sc.close();
    System.out.println();
    System.out.println("Your name is " + name);
    System.out.println("Your age is " + age);
    System.out.println("Your gender is " + gender);
    System.out.println("Your height is " + height);
  }
}
```

Screen output:

```
Enter your name: Tom
Enter your age: 18
Enter your gender: M
Enter your height: 1.72

Your name is Tom
Your age is 18
Your gender is M
Your height is 1.72
```

Class

Concepts of packages will be covered later.

- Both C++ and Java allow class members to have public, protected, or private accessibility
- Java also uses default (package) accessibility when no other accessibility is specified
- The meaning of protected is somewhat different in C++ and Java
  - C++ protected members can only be accessed from derived classes
  - Java protected members can be accessed from derived classes and from other classes in the same package
- C++ breaks the declarations of members up into groups, proceeded by an access specifier followed by a colon, while Java requires each member to have the access specified as part of its declaration
- Initialization of data members in C++ can be done via member initialization list (MIL), but MIL is not available in Java
- Java allows member initialization on declarations
- Member functions in C++ can be made as constant member functions, but this is not available in Java

Constructors

- Both C++ and Java provide a default constructor if you do not define one
- However, Java has no default copy constructor and does not support implicit call of copy constructor (why? Consider the following!)

```
public class Person
{
  private string name = "Tom";
  private char gender = 'M';
  private int age = 18;
  public Person() { }
  // The following shows a similar construct of copy constructor in C++
  public Person(Person p) {
    // ...
  }
  public static void main(String[] args) {
    Person p1 = new Person();
    Person p2 = p1; // What does it mean?
    Person p3(p1);  // How about this? Explicit call of
                    // copy constructor! This is ok
  }
}
```

this Keyword

- In C++, "this" variable is a pointer to the object whose member function is being invoked
  - Syntax: (*this). or this$->$ is used to access members in the same class
- In Java, "this" variable is a reference to the object whose method is being invoked
  - Syntax: this. is used to access members in the same class

Class (C++)

```
// Person-Class.h
#ifndef PERSON_H
#define PERSON_H
#include <iostream>
using namespace std;
class Person {
  // C++ breaks the declarations
  // of members up into groups
  private:
    string name;
    char gender;
    int age;
  public:
    Person::Person(string name,
                   char gender,
                   int age)
        // Initialization of
        // data members using MIL
        : name(name),
          gender(gender),
          age(age) {}
    // Constant member functions exist
    string Person::getName() const
      { return name; }
    char Person::getGender() const
      { return gender; }
    int Person::getAge() const
      { return age; }
    void Person::setName(string name)
      { this->name = name; }  // this pointer
    void Person::setGender(char gender)
      { this->gender = gender; } // this pointer
    void Person::setAge(int age)
      { this->age = age; }    // this pointer
    void Person::print() const {
      cout << "Name: " << name << endl;
      cout << "Gender: " << gender << endl;
      cout << "Age: " << age << endl;
    }
}; // This semicolon is important
#endif
```

Class (Java)

```java
// Person.java
public class Person {
  // Initialization can be done on declarations
  private String name = "Tom";
  private char gender = 'M';
  private int age = 18;
  // No member initialization list
  public Person(String name, char gender, int age) {
    this.name = name;     // this reference
    this.gender = gender; // this reference
    this.age = age;       // this reference
  }
  // No constant methods
  public String getName() { return name; }
  public char getGender() { return gender; }
  public int getAge() { return age; }
  public void setName(String n) { name = n; }
  public void setGender(char g) { gender = g; }
  public void setAge(int a) { age = a; }
  public void print() {
    System.out.println("Name: " + name + "\nGender: " + gender + "\nAge: " + age);
  }
} // No semi-colon here
```

Separation of Class Definition and Implementation

- C++ allows the definition of a class to be declared in a separate file from its implementation
  - Class definition in .h file
  - Class implementation in .cpp file
- Java does not support the kind of separation of class definition and implementation

Separation of Class Definition and Implementation (C++)

```
// Person.h
#ifndef PERSON_H
#define PERSON_H
#include <iostream>
using namespace std;
class Person {
  private:
    string name;
    char gender;
    int age;
  public:
    Person(string name,
           char gender,
           int age);
    string getName() const;
    char getGender() const;
    int getAge() const;
    void setName(string name);
    void setGender(char gender);
    void setAge(int age);
    void print() const;
}; // This semicolon is important
#endif
```

```
// Person.cpp
#include "Person.h"
Person::Person(string name,
               char gender,
               int age)
   : name(name), gender(gender), age(age) {}
string Person::getName() const
  { return name; }
char Person::getGender() const
  { return gender; }
int Person::getAge() const
  { return age; }
void Person::setName(string name)
  { this->name = name; }
void Person::setGender(char gender)
  { this->gender = gender; }
void Person::setAge(int age)
  { this->age = age; }
void Person::print() const {
  cout << "Name: " << name << endl;
  cout << "Gender: " << gender << endl;
  cout << "Age: " << age << endl;
}
```

No Separation of Class Definition and Implementation (Java)

```java
// Person.java
public class Person {
  private String name;
  private char gender;
  private int age;
  public Person(String name, char gender, int age) {
    this.name = name;
    this.gender = gender;
    this.age = age;
  }
  public String getName() { return name; }
  public char getGender() { return gender; }
  public int getAge() { return age; }
  public void setName(String n) { name = n; }
  public void setGender(char g) { gender = g; }
  public void setAge(int a) { age = a; }
  public void print() {
    System.out.println("Name: " + name + "\nGender: " + gender + "\nAge: " + age);
  }
} // No semi-colon here
```

Inheritance

- C++ allows multiple inheritance, i.e. a class may inherit from any number of base classes
- Different from C++, Java allows a class inherits from exactly one base class (Recall, the term superclass is used instead in Java)
  - If no base class is specified, the base class is the class "Object", which thus serves as a root class for the entire inheritance hierarchy
- In C++, base class is specified using colon (i.e. :), but in Java, "extends" keyword is used
- In C++, initialization of data members inherited from base class is done using member initialization list, but in Java, "super" keyword is used at the start of the constructor body to invoke constructor of the base class
- In C++, three kinds of inheritance are available namely public, protected and private. In Java, only public inheritance is available
  - Note: In Java, if a method is public in the base class and you override it, your overridden method must also be public

Inheritance (C++)

```
// Student.h
#ifndef STUDENT_H
#define STUDENT_H

#include "Person.h"

// : is used for inheritance
// Can specify different type of
// inheritance by modifying the
// accessor modifier after :
// This example uses public
// inheritance
class Student : public Person {
  private:
    string major;
  public:
    Student(string name, char gender,
            int age, string major);
    string getMajor() const;
    void setMajor(string major);
    void print() const;
};
#endif
```

```
// Student.cpp
#include "Student.h"
Student::Student(string name, char gender,
                 int age, string major)
  // Initialization of inherited data
  // members is done via
  // member initialization list
  : Person(name, gender, age), major(major)
{}

string Student::getMajor() const {
  return major;
}

void Student::setMajor(string major) {
  this->major = major;
}

void Student::print() const {
  // Call print() of the base class
  Person::print();
  cout << "Major: " << major << endl;
}
```

Inheritance (Java)

```java
// Student.java

// extends is used for inheritance
// No access modifier shoudl be placed after extends
// since only public inheritance is available
public class Student extends Person {
  private String major;

  public Student(String name, char gender, int age, String major) {
    // Initialization of inherited instance variables is done
    // using the keyword super
    super(name, gender, age);
    this.major = major;
  }

  public String getMajor() { return major; }

  public void setMajor(String major) { this.major = major; }

  public void print() {
    // Call print() of the superclass
    super.print();
    System.out.println("Major: " + major);
  }
}
```

Polymorphism: Static and Dynamic Binding (C++)

```
// static-binding.cpp
#include <iostream>
using namespace std;
class A {
  public:
    void funcX() { cout << "funcX in A" << endl; }
    void funcY() { cout << "funcY in A" << endl; }
};

class B : public A {
  public:
    void funcY() { cout << "funcY in B" << endl; }
};
```

- Which version of funcY() will be called by doing the following?

  ```
  A* ptr = new B;
  ptr->funcY();
  ```

  The funcY() in A is called. As funcY() is non-virtual function and so the pointer variable "ptr" is declared to refer to an object of class A, the class A version of funcY() would be used, regardless of the actual type of the object the variable "ptr" currently points to. (Static binding)

Polymorphism: Static and Dynamic Binding (C++)

```
// dynamic-binding.cpp
#include <iostream>
using namespace std;
class A {
  public:
    void funcX() { cout << "funcX in A" << endl; }
    virtual void funcY() { cout << "funcY in A" << endl; }
};

class B : public A {
  public:
    virtual void funcY() { cout << "funcY in B" << endl; }
};
```

- Which version of funcY() will be called by doing the following?

  ```
  A* ptr = new B;
  ptr->funcY();
  ```

  The funcY() in B is called. As funcY() is virtual function and so the pointer variable "ptr" is declared to refer to the actual type of the object the variable "ptr" currently points to. (Dynamic binding)

Polymorphism: Static and Dynamic Binding (Java)

- How about Java?

```java
// DynamicBindingEx.java
class A {
  public void funcX() { System.out.println("funcX in A"); }
  public void funcY() { System.out.println("funcY in A"); }
}

class B extends A {
  @Override
  public void funcY() { System.out.println("funcY in B"); }
}
```

Which version of funcY() will be called by doing the following?

```java
A aRef = new B();
aRef.funcY();
```

The funcY() in B is called as Java does dynamic binding for overriding by default.

How to call funcY in A?

Const-ness

- Both C++ and Java allow a variable declaration to specify that the variable is a constant, i.e. its value cannot be changed after it is declared. C++ uses the keyword "const" for this, while Java uses "final"

C++

```
const double PI = 3.14159;
```

Java

```
final double PI = 3.14159;
```

- There is a major difference between C++ const and Java final when they are applied to functions / methods

C++

```
class A {
  public:
    const Person* func() { ... }
};
```

Java

```
public class A {
  public final Person func()
  { ... }
}
```

  - Left: It means func() returns a pointer to Person, which cannot be used to modify the Person object to which it points
  - Right: It means func() cannot be overridden in any subclass / derived class of the A

C++ Features that are not available in Java

- Preprocessor (e.g. #include, #ifndef, #define, #endif)
- Namespace
- Enumeration types
- Forward declaration
- Named types (i.e. typedef)
- Structure (i.e. struct)
- Scope resolution operator (i.e. ::)
  - Java uses dot operator for everything
- Default arguments
- Implicit type conversion
- Friend functions / classes
- Operator overloading
- Templates (i.e. function templates or class templates)

Features that are available in Java but not in C++

- Interface
- Object class and reflection
- ...

Java Application Programming Interface (API)

- Java Application Programming Interface (API) is a collection of classes with their respective fields, constructors, and respective methods
- The APIs provide information about classes, parameters, and other useful information to programmers

Java 8 API Documentation:
`https://docs.oracle.com/javase/8/docs/api/`

Eclipse: Press <F2>

Useful tip: To show the documentations for the selected type / class / method in Eclipse: <SHIFT-F2>

That's all!

Any questions?

Appendix: List of Java Reserved Words

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used

** added in Java 1.2

*** added in Java 1.4

**** added in Java 1.5

Appendix: Arithmetic, Relational and Logical Operators

Precedence	Operator	Description	Associativity
1 (highest)	() [] .	Parentheses Array subscript Member selection	Left to Right
2	++ −−	Unary post-increment Unary post-decrement	Right to Left
3	++ −− + − ! ∼ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to Left
4	* / %	Multiplication Division Modulus	Left to Right
5	+ −	Addition Subtraction	Left to Right

Appendix: Arithmetic, Relational and Logical Operators

Precedence	Operator	Description	Associativity
6	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to Right
7	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to Right
8	== !=	Relational is equal to Relational is not equal to	Left to Right
9	&	Bitwise AND	Left to Right
10	^	Bitwise exclusive OR	Left to Right
11	|	Bitwise inclusive OR	Left to Right
12	&&	Logical AND	Left to Right
13	||	Logical OR	Left to Right
14	? :	Ternary conditional	Right to left
15 (lowest)	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left