1.

a)

Implicit conversions are widening – there is little or no information lost, e.g.:

int i = 5;

double d = i;

Explicit conversions are narrowing – there is always potential information loss:

double i = 5.0;

int j = (int) i;

b)

Constructors are not inherited by subclasses, but they can be invoked with the super() keyword. If there is no constructor specified, super() is called automatically, but it is with no arguments, so there would be a compilation error if the superclass has no no-args constructor.

c)

They can be shared between threads or data structures without worrying about change, they decrease garbage collection overhead

2.

a)

i)

The code goes through the list of calls, counts the instances of unique numbers and puts them as keys in a pre-made map of (number, total duration) tuples, counts total duration for each phone number, then counts total charge as double the total duration, except for the favourite phone number. The favourite phone number’s charge is the minimum of 100 or the usual rate.

ii)

<Call> is the specification for the type of elements stored in the list. If there was no specification, it would be an Object type and the specific Call methods would not be accessible.

iii)

Since Java is statistically typed, the left side is the declaration of a variable *totals*, which sets space in memory for a Map (the identifier), while also associating the variable with that memory location. The right side initialises the variable with an empty HashMap. The value of the right side must be some form of the identifier on the left.

iv)

The if statement checks whether there is already a phone number in the totals map, and if not, the body puts a (number, 0) tuple in the map. If there was no check, line 7 would not work as “totals.get(call.number)” would return null, which cannot be summed up with the call duration.

v)

.equals() compares values, which is the comparison needed, unlike “==”, which would compare memory locations of the two strings.

b)

i)

The fields i and j are set to 4 because 4 is the argument given to the constructor, which sets those values.

ii)

The value of i did not change as it is an instance field set to the b1 object. The value of j changed as it is a static field in relation to the whole class; hence, every instance’s j would change to -3.

iii)

B.j, as it would be clear it is the same for the whole class

3.

a)

**public** **class** Bicycle {

**private** **int** id;

**private** String type;

**private** **boolean** available;

**public** Bicycle(**int** id, String type) {

**this**.id = id;

**this**.type = type;

**this**.available = **true**;

}

**public** **boolean** isAvailable() {

**return** available;

}

**public** **void** setAvailable(**boolean** available) {

**this**.available = available;

}

**public** **int** getId() {

**return** id;

}

**public** String getType() {

**return** type;

}

}

b)

Bicycle bike = **new** Bicycle(5, "Details");

c)

**import** java.time.Duration;

**import** java.time.Instant;

**public** **class** Customer {

**private** Bicycle bike;

**private** Instant rentStart;

**public** **void** rentBike(Bicycle bike) **throws** IllegalArgumentException{

**if**(this.bike != **null** || !bike.isAvailable()) {

**throw** **new** IllegalArgumentException();

}

**else** {

bike.setAvailable(**false**);

**this**.rentStart = Instant.*now*();

**this**.bike = bike;

}

}

**public** **double** endRental() **throws** IllegalArgumentException {

**if**(**this**.bike == **null**) {

**throw** **new** IllegalArgumentException();

}

**else** {

Instant rentEnd = Instant.*now*();

**long** difference = Duration.*between*(rentEnd, rentStart).toHours();

**return** difference \* 2;

}

}

}

d)

c.rentBike(b);

e)

The field would be a List or a Set of Bicycle objects, rentBike() would check whether the list/set contains the bike; if there were no errors, the bike would be added to the list/set